INTEGER PROGRAMMING

DISSERTATION

SUBMITTED IN PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

Master of Philosophy

IN

STATISTICS

BY

AMIR ASIF

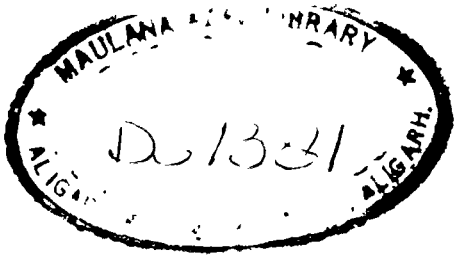
Under the Supervision of

Dr. S. U. KHAN

Professor

DEPARTMENT OF STATISTICS
ALIGARH MUSLIM UNIVERSITY
ALIGARH (INDIA)

1988



To Majaa aunty, my grandmother and the memory of my grandfather

PREFACE

This dissertation entitled, " Integer Programming " is being submitted to the Department of Statistics, A.M.U.,Aligarh, in partial fulfilment for the award of the degree of Master of Philosophy in Statistics.

The intent of this manuscript is to present a survey of the available literature on integer programming. The dissertation concists of five chapters with a comprehensive list of references given at the end. The references are arranged alphabetically according to the author's name.

Chapter I presents a introduction to the survey. It includes definition, mathematical formulation, applications and introduction to methods of integer programming. The next four chapters are algorithm oriented. Chapter II covers cutting plane techniques applicable to integer linear programming. Branch and bound methods and partitioning algorithms are treated in chapter III. Zero-one implicit enumeration is discussed in chapter IV. The closing chapter (chapter V) relates to specialised integer programming problems. It presents a description of the knapsack, fixed-charge

set covering and traveling salesman problems.

I am gratefully indebted to my supervisor Prof. S.U.Khan, Chairman, Department of Statistics, A.M.U.,Aligarh, for untiring supervision, invaluable guidence, constant encouragement, necessary facilities and support provided by him through all the stages of writing this dissertation.

I am also indebted to Prof. S. Rehman (Ex-chairman), Dr. Zaheeruddin, Dr. Abdul Bari and Dr. Mohd. Yaqub for their affections, sympathies, advice and support throughout my studies.

The Department of Statistics, its teachers, my colleagues, friends, younger brother Mr. Abdul Qayyum, and typist Mr. Hafis Alvi, all deserve my sincere thanks for their general support.

Amir Asif

(AMIR ASIF)

Aligarh

September,1988

C O N T E N T S

PAGE NO.

PAGE NO.

PAGE NO.

CHAPTER-I

INTRODUCTION

1.1 DEFINITION OF INTEGER PROGRAMMING

Integer programming deals with the solution of mathematical programming problems in which some or all of the decision variables can assume non-negative integer values only. An integer program is called mixed if some but not all decision variables are restricted to integer values. If all the decision variables are restricted to integer values the integer program is called pure. If in the absence of the integrality conditions the objective and constraint functions are linear, the resulting model is called an integer linear program.

The general integer problem may be defined as :

$$\text{maximise (or minimise)} \quad z = g_0(x_1, x_2, \ldots, x_n)$$

$$\text{subject to} \quad g_i(x_1, x_2, \ldots, x_n) \left\{ \begin{matrix} \leq \\ = \\ \geq \end{matrix} \right\} b_i \quad i \in \{1, 2, \ldots, m\}$$

$$x_j \geq 0 \quad j \in \{1, 2, \ldots, n\}$$

$$x_j \text{ an integer} \quad j \in I \subseteq N$$

If $I = N$, that is, all the variables x_j are restricted

to integer values, the problem is called a pure integer problem. Otherwise, if $I \subset N$, then the problem is called a mixed integer problem.

The integer linear program is written as :

$$\text{maximize(or minimize)} \quad z = \sum_{j=N} c_j x_j$$

$$\text{subject to} \quad \sum_{j \in N} a_{ij} x_j + s_i = b_i, \quad i \in M$$

$$s_i \geq 0 \quad i \in M$$

$$x_j \geq 0 \quad j \in N$$

$$x_j \text{ an integer } j \in I \subset N$$

where s_i is a slack variable.

It may appear that the additional integrality condition should not present a serious problem. Although the solution space of the integer problem is structurally better defined than in the continuous problem, it has proved to be computationally formidable. To date, inspite of three decades of continuous theoretical research, together with a tremendous increase in the speed and power of the digital computer, the developed integer algorithms have not yielded satisfactory computational results. The fact of the matter is that the

integrality condition often destroys the " nice " properties of the solution space. A typical example is the integer linear problem. In the absence of the integer condition, the solution space is convex. This is primarily the basic property that leads to the tremendously successful simplex method for solving linear program.

Mixed integer (or integer) programs in which the integer variables are constrained to be 0 or 1 are called zero-one mixed integer (or integer) programs.

A nonlinear function can also be linearized by first transforming it into a polynomial function with zero-one variables (Hammer and Rudeanu [1968]) and then transforming the polynomial function into a linear function with zero-one variables (Watter [1967], Zangwill [1965]). However, these transformations tend to dramatically increase the number of variables and constraints Glover [1972], and Glover and Wooley [1975], [1975] have discussed methods to achieve more economical linear representations of zero-one polynomial programming problems. A computational study by Taha [1970] indicates that the benefit gained by using these transformations is data

dependent, and conversion may not be worthwhile. Related work includes the relationship between a zero-one integer program and a quadratic programming problem as discussed in Bowman and Glover [1972], Kennington and Fyffe [1972], and Raghavachari [1969], [1970], other transformations are in Garfinkel and Nemhauser [1972], Granot and Hammer [1972], Peterson [1971], Balas and Mazzola [1980], [1984], and Lu and William [1987].

1.2 APPLICATIONS OF INTEGER PROGRAMMING

The importance of integer programming (mixed or pure) is well established. A number of mathematical programs can be converted to problems with integer variables. Many situations yield programming formulations with some or all the variables required to be integer. Scheduling, location, network and selection problems which appear in industry, military, education, health and other environments are also formulated as integer programming problems. A complete discussion on classical models may be found in Balinski [1965], Balinski and Spielberg [196], and Dantzig [1960]. A few integer programming formulations may represent a vast number of real world problems. These problems have been titled and algorithms have been developed. Facility

location problems (plant location problems), resource-task scheduling (set covering and set partitioning problems), a loading problem (Knapsack problem) and traveling salesman problem are the most popular cases. These problems have been formulated as integer programming problems. Mathematical models, algorithms and applications of these problems are left to Chapter V.

1.3 METHODS OF INTEGER PROGRAMMING

Dantzig [1949] showed that certain integer programs may be solved as linear ones. Well-known cases include the assignment, transportation, and static max flow problem. Most integer programs will not exhibit this integrality property, and the simplex method will generally not solve mixed integer (or integer) programs. Some believe that it may be better to solve the problem as linear one and round the results " intelligently" to acquire a good (not optimal) solution quickly. The idea of rounding is not without merit, at least for the time being and specially when one is confronted with having either " a " solution or no solution at all. However, one must be aware of the (sometimes severe) limitations of rounding. If a feasible

solution is obtained by rounding, one should not be under the illusion that such a solution is optimal or even close to optimal. The fact that continuous optimization is used does not necessarily mean that it will lead to a " good " integer solution. The rounding procedure at best may be regarded as a heuristic.

Any integer model having an original equality constraint can never yield a feasible integer solution through rounding. This is based on the assumption that only basic variables can be rounded, if necessary, and that all the nonbasic variables remain at zero level. The assumption is not unreasonable since it is generally difficult to consider elevating a nonbasic variable above zero while maintaining feasibility. This result was observed by Glover and Sommer [1972]. The idea of rounding in which the nonbasic variables are allowed to assume positive (integer) values serves as the basis for developing a sophisticated, but still computationally difficult, method for solving integer plinear programming problems.

The main conclusion from the above discussion is that the use of rounding is not a reliable procedure for dealing with

the general integer problem. This emphasised the importance of continued research in order to develop efficient algorithms for the integer problem.

A classification of the methods for solving integer programming problem is given in the following fig.1.1

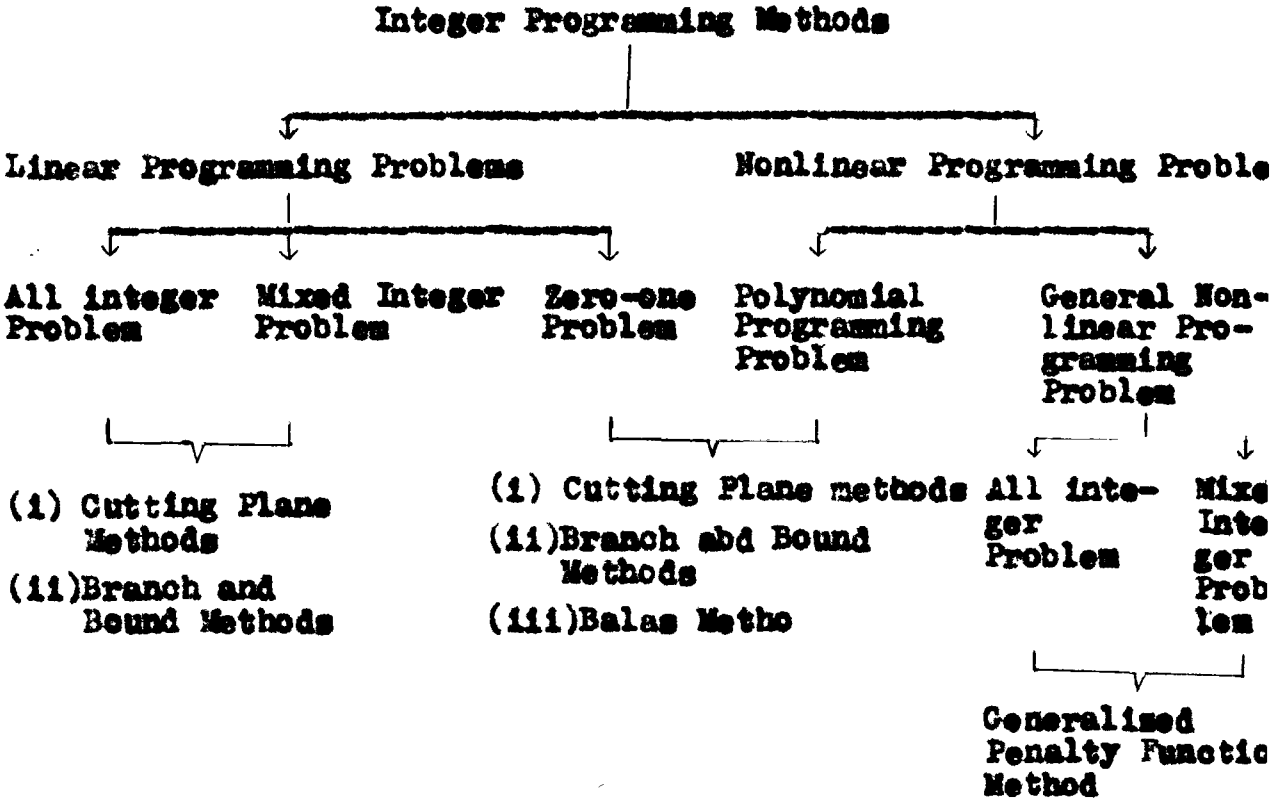


Figure 1.1

We shall consider principal approach for solving integer programs, which are generally categorized into three types :

(i) enumerative methods (ii) cutting plane methods and (iii) partitioning algorithms. We briefly describe these methods. The details of algorithms are left to the subsequent chapters.

(i) Enumerative Methods :

Enumerative methods seek enumerating all the finite number of points of the solution space. In the obvious sense this statement may seem ambiguous, if not erroneous, when applied to the mixed integer problem. Technically, however, it is shown later that, even in the mixed case, the enumerative methods are primarily controlled by the integer variables. What makes enumerative methods more promising than simple exhaustive enumeration, however, is that techniques can be developed to enumerate only a portion (hopefully small) of all the candidate solutions while automatically discarding the remaining points as non promising. Clearly, the efficiency of the resulting search algorithm depends on the power of the techniques that are developed to discard the non-promising points.

Enumerative methods primarily include implicit enumeration techniques and branch-and-bound techniques. The first type is

mostly suited for the zero-one problem, and may actually be considered as a special case of the branch and bound methods. A detail description of these methods with computational results is given in Chapters III and IV.

(ii) Cutting Plane Methods :

Cutting plane algorithms are developed primarily for the (mixed or pure) integer linear program. These algorithms are motivated by the fact that the simplex solution to a linear program must occur at an extreme point. The intent is then to add specially deduced supplementary inequalities that are violated by the current noninteger solution but never by an feasible (integer) point. The successive application of such a procedure should eventually result in a new (convex) solution space with its optimum extreme point properly satisfying the integrality condition. The name " cutting " methods is given by the fact that the supplementary inequalities (constraints) " cut " off infeasible parts of the continuous solution space.

Another method for the integer problem is inspired by the cutting plane techniques and calls for identifying the convex hull of all the feasible integer points by constructing a set

of proper linear inequalities. Once this is done, the application of the regular simplex method would clearly produce the desired result. The underlying development is usually referred to as the asymptotic problem. A common property that characterises almost all the cutting plane algorithms is that there is no available information about the integer solution if the calculations are stopped prematurely. These methods are usually known as dual techniques and their property represents a major disadvantage. There has been some effort to develop " primal " algorithms, but from the computational viewpoint, these algorithms have been less satisfactory.

The enumerative methods are sometimes referred to as "almost dual " techniques. Although primal feasibility is not guaranteed at all times, occasionally the algorithms may produce a good feasible solution. This is an important advantage as compared with the cutting methods. However, the enumerative methods (specailly the branch and bound type) usually result in severe taxation of the computer memory, a difficulty not present in the cutting methods.

Cutting and enumerative methods possess mutually exclusive

properties which, if combined, may result in a superior method of solution, namely, an algorithm that yields feasible solutions and in the meantime does not require a large computer memory. Various cutting plane algorithms with numerical examples are presented in chapter II.

(iii) Partitioning Algorithms :

A mixed integer program can be posed as an integer problem (Benders [1962]). The difficulty in solving the integer formulation arises out of the large number of constraints which must be generated. To overcome this difficulty Benders proposed a " partitioning algorithm " which solves the integer equivalent of the mixed problem by solving a series of related integer and linear programs. Partitioning algorithms are discussed in chapter III.

1.4 SPECIALIZED ALGORITHMS

There are several specialised methods for solving structured problems such as the Knapsack problem, the fixed charge problem, the set covering problem, and the traveling salesman problem. These algorithms, although designed specifically to exploit the special structure of the problem, are actually in the sprit of either the search methods or the cutting plane methods or a combination of both. The specialised programs are given in chapter V.

CHAPTER-II

CUTTING PLANE TECHNIQUES

2.1 INTRODUCTION

A cut is a derived constraint which has the desirable property of " cutting off " part of the set of feasible solutions while not excluding any integer solutions. Most methods which employ cuts generate cut constraints and utilize them using linear programming methods until an optimal solution is obtained.

The idea of constraint generation was proposed by Dantzig, Fulkerson, and Johnson [1954] in their work on the traveling salesman problem, and then by Markowitz and Manne [1957]. However, Gomory [1958] developed the first cutting plane algorithm applicable to any integer program. Shortly afterward, Gomory [1960] and Beale [1958] generalized Gomory's results to the mixed integer case. Gomory [1960] developed a second cutting plane algorithm for the integer program which requires only additions and subtractions in computation (an " all-integer " technique). All of the above methods maintain linear programs which are dual feasible, and are therefore often classified as dual cutting plane algorithms. Dantzig [1959] proposed a cut for which Gomory and Hoffman [1963] showed that the basic approach may not converge for large classes

of problems. The general difficulties of cutting plane algirithms are described by Halfin [1972], Jeroslow [1969], [1971], Jeroslow and Kortanek [1969], [1971] and Robin D. [1970]. Salkin H. M. [1971] proved that Gomory's dual fractional cut for mixed integer linear programming (MILP) problem applied to an all integer linear programming (AILP) problem gives faster convergence than that of Gomory's dual fractional cut for all integer linear programming (AILP) problems. Ben-Israel and Charnes [1962] developed algorithms which lacked finiteness proof. Young [1968] improved the work of Ben-Israel and Charnes by the help of Glover [1968]. Glover [1968] and Young [1971] developed cutting plane algorithms for the integer programs which contain linear problems that are primal feasible. Balas [1969], Burdet [1970] and Glover [1973] again developed different cutting plane methods for the solution of integer programs. These cuts are basically generated through the intersection of feasible region with a specially structured region. Mathis S. J. [1971] described the development of a counter example to the rudimentary primal integer programming algorithm. He gave two examples. The first showed the particular sequence of cuts that leads to a cycling process. Then, utilizing the algebraic

implications of this sequence of cuts, a second example is developed that does not converge using the usual column selection rule. Arnold,L.[1974] presented a paper in which he examines the sequences of Glover's results using simplified primal integer algorithm so that both the termination of a sequence and the tableau at that point can be predicted. He constructs an algorithm that exploits this structure by performing the iterations of the sequence implicitly and presents computational results. Arnold,L. [1974] presented a second paper in which he develops a new cutting plane for primal integer programming. This cut, when it exists and is used as pivot row, has the property that the minimum of the simplex evaluators of the next tableau is strictly larger than the minimum for the current tableau. The process of cut generation is then imbedded into a convergent primal algorithm.

Recently few more papers have been contributed to the cutting plane techniques. Charnes, A, Granot, D., and Granot, F. [1977] developed a cutting plane algorithm for solving integer interval linear programming problems using the concept of intersection cut, introduced by Balas, E. [1971]. Ghandforoush,P. and Austin, L.M. [1981] developed a primal-dual cutting plane algorithm

for all integer programming. This algorithm is a hybrid (i.e. primal-dual) cutting plane method which alternates between a primal-feasible stage related to Young's simplified primal algorithm, and a dual feasible stage related to Gomory's dual all-integer algorithm. Austin, L.M. and Ghandforoush, P. [1983] developed an advanced dual algorithm with constraint relaxation for all integer programming. In this algorithm they generate upper bounds on the decision variables, and use the bounds to create an advanced starting point for a dual all integer cutting plane algorithm. In addition, they use a constraint derived from the objective function to speed progress toward the optimal solution. Their basic vehicle is the dual all-integer algorithm of Gomory, but they incorporate certain row-and-column selection criteria which partially avoid the problem of dual-degenerate iterations.

In this chapter, we present all the basic cutting plane algorithms with solved examples. For recent work the references are given.

2.2 DANTZIG'S CUT : Dantzig [1959] observed that, given a

non-integer program, a legitimate cut constraint is that the sum of the nonbasic variables must equal or exceed one.

$$\sum_{j \in J} x_j \geq 1$$

where J is the set of indices corresponding to the non-basic variables appearing in the optimal LP solution.

The justification is obvious: Under no circumstance could the sum of the non-basic variables be negative. Since all non-basic variables equal to zero (which is the noninteger optimal) was not an integer solution, at least one non-basic variable must equal or exceed one, which is exactly Dantzig's constraint.

2.2.1 Dantzig's Algorithm : A procedure based on Dantzig's cut was suggested :

Step I - Solve the linear programming problem, ignoring the integer constraints.

Step II- If the optimal solution is integer, stop. Otherwise go to step III.

Step III- Add a constraint requiring that the sum of non-basic variables be at least one.

Step IV- Use the dual simplex method to solve the altered problem. Then go to Step II.

2.2.2 Limitation of Dantzig's Cut : Although this algorithm does converge to an optimum in a finite number of iterations in many instances, it has been proved by Gomory and Hoffman [1963] that it does not work in general.

A variation of the Gomory all-integer algorithm may be viewed as an adaptation of Dantzig's method that does converge in every case.

2.3 GOMORY'S CUTS : Ralph Gomory has made many outstanding contributions to the field of integer programming. We shall consider four of his cuts.

2.3.1 Gomory's fractional Cut : The classical dual fractional (convergent) algorithm for integer program was developed by Ralph Gomory [1958]. Gomory's cut has the form

$$x_{n+m+k} = -f_{vo} + \sum_{j=1}^{n} (-f_{vj})(-x_{J(j)}) \geq 0$$

where x_{n+m+k} is the " Gomory stack variable " associated with the (kth) added inequality and $f_{vj} = a_{vj} - |a_{vj}|$ (j=o,1,...n).

Observe that $0 < f_{vo} < 1$, $0 \leq f_{vj} \leq 1$ $(j=1,\ldots,n)$, and when the inequality is appended to the linear optimal tableau, primal infeasibility is introduced, since $x_{m+n+1} = f_{vo} < 0$. It has also been shown that the Gomory cut is satisfied by every integer solution and that it has properties sufficient to support a finite algorithm.

2.3.2 Gomory's fractional Algorithm : Gomory's fractional algorithm [1963] may be described by the following sequence of steps :

Step I- Starting with an all integer tableau, solve the integer program as a linear one. If it is infeasible, so the integer problem terminate. If the optimal solution is all integer, the integer program is solved - terminate. Otherwise, go to Step II.

Step II- Derive a new inequality constraint (or cut) from the integrality and (current) constraint requirements which "cuts off" the current optimal point (i.e. makes the linear programming solution infeasible) but does not eleminate any integer solution. Add the new inequality to the bottom of the simplex tableau which then exhibits primal infeasibility (The value of the slack variable associated with the new row will be negative).

Go to step III.

Step III- Reoptimize using the dual simplex method. If the new linear program is infeasible the integer problem has no solution - terminate. If the new optimum is in integer, the integer program is solved - terminate. Otherwise, go to step II.

his framework is rather simple and , except for the nature of the cut constraint in step II, identical to the framework of Dantzig's method.

Convergence of Gomory's fractional Algorithm

In order to prove convergence of Gomory's fractional algorithm, we must alter the algorithm slightly and make a few assumptions. We assume that every problem variable, including the objective function variable, has a lower bound.(We could, if desired, use linear programming formulations to find such bounds). Next we assume that the non-integer optimum is such that all non-basic vectors are lexicographically positive. A vector is said to be lexicographically positive (negative) if its first non-zero component is positive (negative).

2.3.3 **Numerical Example** : Maximise $-x_1 - x_2 = x_0$

Subject to $-4x_1 - 10x_2 \leq -12$

$-10x_1 - 4x_2 \leq -12$

$x_1, x_2 \geq 0$, integer

With non-negative slack x_3 and x_4, the linear optimal simplex tableau appears below and is obtained after two pivots.

Tableau-1

	1	$-x_4$	$-x_3$
x_0	-144/84	6/84	6/84
$-x_1$	72/84	-10/84	4/84
x_2	72/84	4/84	-10/84
x_3	0	0	-1
x_4	0	-1	0
x_5	-72/84	(-74/84)	-4/84

$(x_1 = x_2 = 72/84,\ x_0 = 12/7)$

$a_{10} = 72/84$

$f_{10} = \frac{72}{84} - \left[\frac{72}{84}\right] = \frac{72}{84}$

$a_{11} = -10/84$

$f_{11} = -10/84 - [-10/84] = 74/84$

$a_{12} = 4/84 - [9/84] = 4/84$

Pivot column selection : Min $\left\{\frac{6/84}{|-74/84|}, \frac{6/84}{|-4/84|}\right\} = \frac{6}{74}$

$\therefore$ x_4 becomes basic.

Adding the constraint x_5 while x_1 is selected as source row,

on pivoting we obtain simplex tableau 2.

Tableau-2

	1	$-x_5$	$-x_3$	(x_1=72/74, x_2=60/74, x_0= -132/74)
x_0	-132/74	6/74	5/74	a_{20} = 60/74
x_1	72/74	-10/74	4/74	f_{20} = 60/74-[60/74]=60/74
x_2	60/74	4/74	-9/74	a_{21} = 4/74
x_3	0	0	-1	f_{21} = 4/74-[4/74]= 4/74
x_4	72/74	-84/74	4/74	f_{22} = -9/74-[-9/74]= 65/74
x_5	0	-1	0	
x_6	-60/74	-4/74	(-65/74)	

Pivot column selection, Min $\left\{ \dfrac{6/74}{|-4/74|}, \dfrac{5/74}{|-65/74|} \right\} = \dfrac{5}{65}$

$\therefore$ x_3 = becomes basic.

x_1 is selected as source row. Adding x_6 constraint to the bottom tableau 2. On pivoting we obtain tableau 3.

Tableau-3

	1	$-x_5$	$-x_6$	$(x_1=x_2=60/65,\ x_0=-120/65)$
$-x_0$	-12/65	5/65	5/65	$a_{00} = -120/65$
x_1	60/65	-9/65	4/65	$f_{00} = -120/65 - \left[\frac{-120}{65}\right] = 10/65$
x_2	60/65	4/65	-9/65	$a_{01} = 5/65$
x_3	60/65	4/65	-74/65	$f_{01} = 5/65 - [5/65] = 5/65$
x_4	60/65	-74/65	4/65	$a_{02} = 5/65$
x_5	0	-1	0	$f_{02} = 5/65 - [5/65] = 5/65$
x_6	0	0	-1	
x_7	-10/65	(-5/65)	-5/65	

Pivot column selection : Min $\left\{\frac{5/65}{|-5/65|}, \frac{5/65}{|-5/65|}\right\}$, tie

Compute, Min $\left\{\frac{-9/65}{|-5/65|}, \frac{4/65}{|-5/65|}\right\} = -9/65$

$\therefore$ x_5 becomes basic.

Tableau-4

	1	$-x_7$	$-x_6$
x_0	-2	1	0
x_1	6/5	-9/5	1/5
x_2	4/5	4/5	-1/5
x_3	4/5	4/5	-6/5
x_4	16/5	-74/5	6/5
x_5	2	-13	1
x_6	0	0	-1
x_7	0	-1	0
x_8	-1/5	-1/5	(-1/5)

($x_1 = 6/5$, $x_2 = 4/5$, $x_0 = -2$)

$a_{10} = 6/5$

$f_{10} = 6/5 - [6/5] = 1/5$

$a_{11} = -9/5$

$f_{11} = -9/5 - [-9/5] = 1/5$

$a_{12} = 1/5$

$f_{12} = 1/5 - [1/5] = 1/5$

Pivot column selection :

$$\text{Min} \left\{ \frac{1}{|-1/5|}, \frac{0}{|-1/5|} \right\} = 0$$

$\therefore x_6$ becomes basic.

Tableau-5

	1	$-x_7$	$-x_8$
x_0	-2	1	0
x_1	1	-2	1
x_2	1	1	-1
x_3	2	2	-6
x_4	2	-16	6
x_5	1	-14	5
x_6	1	1	-5
x_7	0	-1	0
x_8	0	0	-1

($x_0 = -2$ with $x_1 = x_2 = 1$)

All integer optimal tableau.

The inequalities $x_5, x_6, x_7, x_8 \geq 0$ in terms of x_1 and x_2 are :

$$x_5 = -12 + 9x_1 + 4x_2 \geq 0$$

$$x_6 = -12 + 4x_1 + 9x_2 \geq 0$$

$$x_7 = -2 + x_1 + x_2 \geq 0$$

$$x_8 = -3 + x_1 + 2x_2 \geq 0$$

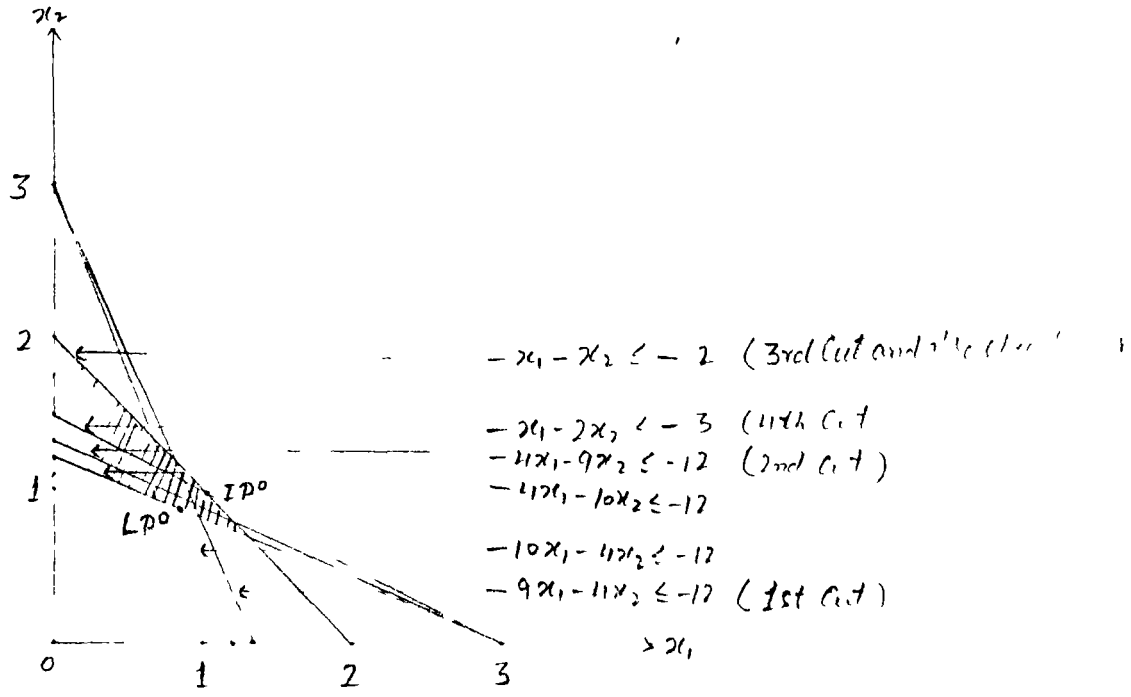


Figure 2.1 - A graph of the Numerical Example 2.3.3 showing solutions, constraints, and cut constraints.

2.3.4 Gomory's Fractional Mixed Integer Cut : Ralph Gomory [1960] developed cutting plane algorithm for the mixed integer program which is a direct extension of the fractional integer programming algorithm. The algorithm may be considered as a mixed integer, primal dual feasible starting solution, cut method. It is similar to the fractional algorithm but, in this case, not all the variables are required to be integral. The cut has the form

$$x_{n+m+k} = -f_{vo} + \sum_{j=1}^{n} (-g_{vj})\,(-x_{J(j)}) \geq 0$$

where x_{n+m+k} is the "Gomory's Slack Variable " associated with the (kth) added inequality.

$$f_{vj} = a_{vj} - [\,a_{vj}\,] \qquad (j = o,1,\ldots,n)$$

$$g_{vj} = \begin{cases} a_{vj} & \text{if } a_{vj} \geq o \text{ and } x_{J(j)} \text{is a continuous va} \\ \dfrac{f_{vo}}{f_{vo}-1} \cdot a_{vj} & \text{if } a_{vj} < o \text{ and } x_{J(j)} \text{is a continuous var} \\ f_{vj} & \text{if } f_{vj} \leq f_{vo} \text{and } x_{J(j)} \text{is an continuous var} \\ \dfrac{f_{vo}}{1-f_{vo}}\,(1-f_{vj}) & \text{if } f_{vj} > f_{vo} \text{ and } x_{J(j)} \text{is an integer var} \end{cases}$$

Observe that $o < f_{vo} < 1$ and thus when the inequality is

added to the bottom of the linear optimal tableau, primal infeasibility is introduced, since $x_{n+m+k} = -f_{vo} < o$. It can be shown that the Gomory mixed integer cut is implied by the source row and integrality requirements, and that it has properties which guarantee a finite algorithm.

2.3.5 Gomory's Fractional Mixed Integer Algorithm : This algorithm may be described by the following sequence of steps :

Step I- Solve the mixed integer program as a linear one. If it is infeasible, so is the mixed integer problem - terminate. If the optimal solution is integer in the integer constrained variables, the mixed integer program is solved - terminate. Otherwise, go to step II.

Step II- From a row corresponding to an integer constrained variable which does not have an integral value, derive a new inequality constraint which " cuts off " the current optimal point but does not eleminate any mixed integer solution. Add the new inequality to the bottom of the simplex tableau which then exhibits primal infeasibility. (The slack variable associated with the new row will be negative.) Go to step III.

Step III- Reoptimise using the dual (lexicographic) simplex method. If the new linear program is infeasible (the dual will be unbounded), the mixed integer problem has no solution- terminate. If the new optimal solution is integer in the integer constrained variables, the mixed integer program is solved - terminate. Otherwise, go to step II.

The slack of the cut constraint is not required to be integer. Otherwise, the mixed-integer algorithm proceeds exactly as does the fractional algorithm.

2.3.6 Numerical Example :

			Or	
Maximise	$5x_1 + 2x_2 = x_0$		Min	$-5x_1 - 2x_2 = x_0$
Subject to	$2x_1 + 2x_2 \leq 9$		Subject to	$-2x_1 - 2x_2 \geq -9$
	$3x_1 + x_2 \leq 11$			$-3x_1 - x_2 \geq -11$
	$x_1 , x_2 \geq 0$			$x_1 , x_2 \geq 0$
	x_0 , x_1 integer			x_0 , x_1 integer

With non-negative slack x_3 and x_4, the linear optimal simplex tableau appears below and is obtained after two pivots.

Tableau-1

	1	x_4	x_5
x_0	$18\frac{3}{4}$	3/2	1/4
→ x_1	13/4	1/2	-1/4
x_2	5/4	-1/2	3/4
x_3	0	0	-1
x_4	0	-1	0
x_5	-1/4	-1/2	(-1/12)

$(x_1=13/4,\ x_2=3/4,\ x_0=18\frac{3}{4})$

Linear Optimal tableau

$a_{10}= 13/4 \quad \therefore f_{10}= 13/4-[13/3]=1$

$a_{11}=1/2 \quad \therefore g_{11}= a_{11}= 1/2$

$a_{12}= -1/4$

$$\therefore g_{12}= \frac{f_{10}}{f_{10}-1} a_{12}= \frac{1/4}{1/4-1}\left(-\frac{1}{4}\right)= -\frac{1/4}{-3/4}\cdot\frac{1}{4}= \frac{1}{12}$$

Pivot column selection;

$$\text{Min}\left\{\frac{3/2}{|-1/2|}, \frac{1/4}{|-1/4|}\right\} = \text{Min}\{3,1\} = 1$$

On Pivoting we obtain tableau 2 .

Tableau-2

	1	x_4	x_5
x_0	18	0	3
x_1	4	2	-3
→ x_2	-1	(-5)	9
x_3	3	6	-12
x_4	0	-1	0
x_5	0	0	-1

Pivot Column Selection;

$$\text{Min}\left\{\frac{0}{|-5|}, \frac{3}{9}\right\} = 0$$

Tableau 2 exhibits dual feasibility. On pivoting we get tableau 3.

Tableau-3

	1	x_2	x_5
x_0	18	0	3
→ x_1	18/5	2/5	3/5
x_2	0	-1	0
x_3	9/5	6/5	-6/5
x_4	1/5	-1/5	-9/5
x_5	0	0	-1
x_6	-3/5	(-2/5)	-3/5

$$a_{10} = \frac{18}{5}, \quad f_{10} = \frac{18}{5} - \left[\frac{18}{5}\right] = \frac{3}{5}$$

$$a_{11} = \frac{2}{5} \quad \therefore g_{11} = a_{11} = \frac{2}{5}$$

$$a_{12} = \frac{3}{5} \quad \therefore g_{12} = a_{12} = \frac{3}{5}$$

Pivot column selection

$$\text{Min}\left\{\frac{0}{|-2/5|}, \frac{3}{|-3/5|}\right\} = 0$$

On pivoting we obtain tableau 4.

Tableau-4

	1	x_6	x_5
x_0	18	0	3
x_1	3	1	0
x_2	3/2	-5/2	3/2
x_3	0	3	-3
x_4	1/2	-1/2	-3/2
x_5	0	0	-1
x_6	0	-1	0

$(x_1 = 3, x_2 = 3/2, x_0 = 18)$

Optimal mixed integer solution

The inequalities x_5 and x_6 in terms of x_1 and x_2 are

$$x_5 = 18 - 5x_1 - 2x_2 \geq 0 \quad \text{or} \quad 5x_1 + 2x_2 \geq 18$$

$$x_6 = 3 - x_1 \geq 0 \quad \text{or} \quad x_1 \leq 3$$

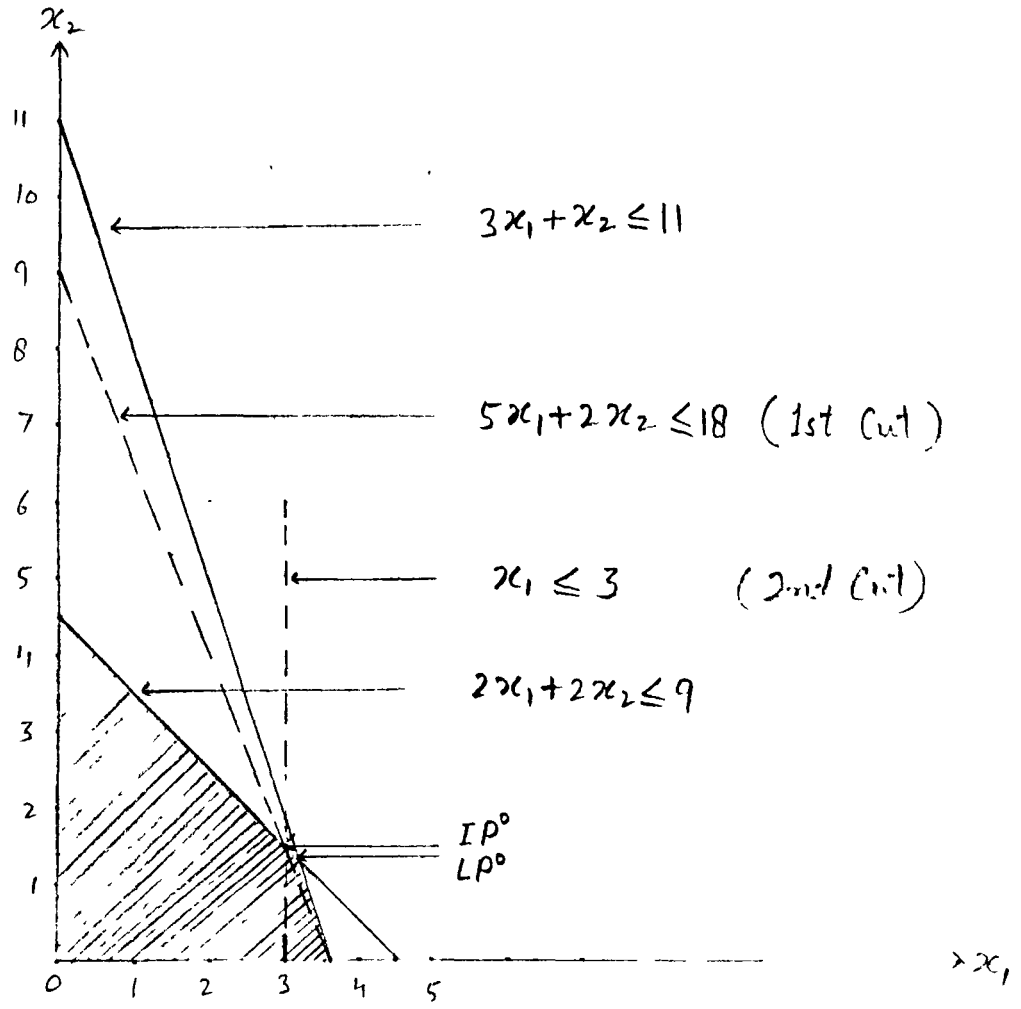


Figure 2.2- A graph of the Numerical Example 2.3.5 showing solutions, constraints, and cut constraints.

2.3.7 Gomory's All Integer Cut : The cutting plane algorithm for an all integer program was developed by Ralph Gomory in 1960. Its similarity to the fractional method is principally due to the utilisation of the lexicographic dual simplex method and to the maintenance of lexicographic positive column in the tableau. However, the basic approach is different from the fractional technique. There is no optimisation, generating of a constraint, reoptimisation, etc. Rather inequalities are generated at each iteration starting with the very first. Furthermore each of these constraints is used as pivot row, and is constructed so that it has integral coefficients and the pivot is -1. The initial tableau is assumed to be all integer and lexicographic dual feasible. Hence, successive tableaux are also all integer and lexicographic dual feasible. The primal integer solution proceeds towards feasibility, and since dual feasibility is maintained, optimality is reached when it is attained.

Note that the method is a direct extension of the classical dual simplex algorithm. The essential difference is that the pivot row in the all integer algorithm is generated at each iteration and ensures a -1 pivot. Since the technique employs

the dual simplex method and maintains all integer tableaux, it is referred to as "dual all-integer ". The cut has the form :

$$x' = \left| \frac{a_{vo}}{\lambda} \right| + \sum_{j=1}^{n} \left| \frac{a_{vj}}{\lambda} \right| (-x_{J(j)}) \geq 0$$

where x' is a non-negative Gomory slack variable and λ is a positive number found by the rules below :

2.3.8 The Rules For Finding λ :

Step 1. With v as the generating row, let α_p be the lexicographically smallest column among those having $a_{vj} < o$ $(j = 1, \ldots, n)$.

Step 2. Let $U_p = 1$, and for every $j \geq 1$ $(j \neq p)$ with $a_{vj} < o$ let U_j be the largest integer such that

$$\left(\frac{1}{U_j} \right) \alpha_j > \alpha_p$$

Step 3. For each $a_{vj} < o$ $(j \geq 1)$, set $\lambda_j = - a_{vj}/u_j$. Note that λ_j is not necessarily an integer.

Step 4. Set $\lambda =$ maximum λ_j. Note that $\lambda \geq \lambda_p = -a_{vp}/u_p \geq 1$, since $u_p = 1$ and $-a_{vp}$ is a positive integer.

The rules for finding λ, and hence the cut, may seem somewhat complicated. However, we can show that inequality (cut) has the properties described previously. In particular, note that $[a_{vo}/\lambda] < o$ or the generated row is a legitimate pivot row. Also observe that the pivot element $[a_{vp}/\lambda] = -1$, since $\lambda \geq \lambda_p = -a_{vp}$. Furthermore, we shall prove that if row v corresponds to the first primal infeasible row every finite number of iterations, the algorithm converges to an optimal integer point.

2.3.9 Algorithm :

Step I- Start with an all integer simplex tableau which contains a lexicographic dual feasible, solution. Go to step II.

Step II- Select a primal infeasible row v(i.e. $a_{vo} < o$, $v \neq o$). If none exist, the tableau exhibits the optimal integer solution -terminate. Go to step III.

Step III- Designate the pivot column a_p ($p = 1,\ldots,n$) to be the lexicographically smallest among those having $a_{vj} < o$. If none exist (i.e. $a_{vj} \geq o$ for $j = 1,\ldots, n$), there is no integer feasible solution-terminate. Go to step IV.

Step IV- Derive an all integer inequality from row v which is not satisfied at the current primal solution. (Its slack will be negative) It must also have a -1 coefficient in column a_p. Adjoint it to the bottom of the tableau and level it the pivot row. Perform a dual simplex pivot operation and return to Step II.

2.3.10 Numerical Example : Solve the following integer program by all integer algorithm

$$\text{Minimise } x_1 + x_2 = x_0$$

$$\text{Subject to } 4x_1 + 10x_2 \geq 12$$

$$10x_1 + 4x_2 \geq 12$$

$$x_1, x_2 \geq 0 \text{ integer}$$

Changing the problem into maximization

$$\text{Maximise } -x_1 - x_2 = x_0^*$$

$$\text{Subject to } -4x_1 - 10x_2 \leq -12$$

$$-10x_1 - 4x_2 \leq -12$$

$$x_1, x_2 \geq 0, \text{ integer}$$

With non-negative slack variables x_3 and x_4, we get

tableau 1

Tableau-1

		1	$-x_1$	$-x_2$
	x_0^*	0	1	1
	x_1	0	-1	0
	x_2	0	0	-1
	x_3	-12	-4	-10
→	x_4	-12	-10	-4
	x_5	-2	(-1)	-1

$(x_1 = x_2 = 0)$

$p = 1$, $(\frac{1}{a_2})\, a_2 \not> a_1$

$a_1 = 1$

$a_2 = -1$, $(\frac{1}{a_2})1 \geq 1$

$\lambda_1 = \frac{10}{1} = 10$

$\lambda_2 = \frac{4}{-1} = -4$

$\lambda = \max \lambda_j = 10$

The derived inequality is

$$x_5 = [\frac{-12}{10}] + [\frac{-10}{10}](-x_1) + [\frac{-4}{10}](-x_2) \geq 0$$

or $x_5 = -2 + (-1)(-x_1) + (-1)(-x_2) \geq 0$

On pivoting we get tableau 2

Tableau-2

	1	$-x_5$	$-x_2$
x_0^*	-2	1	0
x_1	2	-1	1
x_2	0	0	-1
$-x_3$	-4	-4	-6
x_4	8	-10	6
x_5	0	-1	0
x_6	-1	-1	(-1)

$(x_1=2,\ x_2=0,\ x_0^* = -2)$

$p = 2$

$u_2=1$

$u_1=1$

$\lambda_1 = \frac{4}{1} = 4$

$\lambda_2 = \frac{6}{1} = 6$

$\lambda = \max(\lambda_j) = 6$

$(-\frac{1}{u_1})1 \geq 0$

The derived inequality is

$$x_6 = [\frac{-4}{6}] + [\frac{-4}{6}](-x_5) + [\frac{-6}{6}](-x_2) \geq 0$$

or $x_6 = -1 + (-1)(-x_5) + (-1)(-x_2) \geq 0$

On pivoting we get tableau 3

Tableau-3

	1	$-x_5$	$-x_6$
x_0^*	-2	1	0
x_1	1	-2	1
x_2	1	1	-1
x_3	2	2	-6
x_4	2	-16	6
x_5	0	-1	0
x_6	0	0	-1

($x_0^* = -2$ with $x_1 = x_2 = 1$)

or

($x_0 = 2$ with $x_1 = x_2 = 1$)

All integer optimal solution

The derived inequalities $x_5, x_6 \geq 0$ in terms of x_1 and x_2 are

$$x_5 = -2 + x_1 + x_2 \geq 0$$

$$x_6 = -3 + x_1 + 2x_2 \geq 0$$

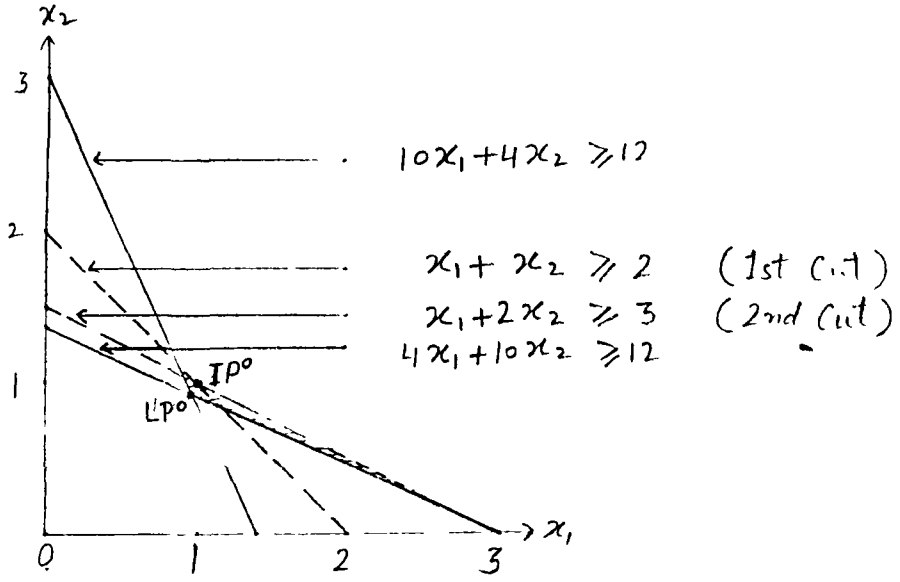


Figure 2.3- A graph of the Numerical Example 2.3.9 showing solutions, constraints, and cut constraints.

To show that the algorithm is finite we assume that x_0, the objective function is bounded below. Then, as in the proof of the fractional algorithm, we suppose that the process is not finite and appeal to the lexicographic decreasing nature of the o column to obtain a contradiction.

If the algorithm is not finite there exists an infinite sequence of tableaux indexed by k, in which the o column is decreasing lexicographically. That is

$$\alpha_o^k \succ \alpha_o^{k+1} \succ \alpha_o^{k+2}, \text{ etc.}$$

As each component of α_o is integer we have a_{oo}, its first component, decreasing by integer amounts. Thus, to remain above its lower bound it must eventually remain fixed. Then, for the o column to continue decreasing lexicographically, a_{1o}, its second component, must decrease. We show that it must eventually remain fixed at a non-negative integer. For suppose a_{1o} falls below o; then now 1 becomes an eligible source row. If it is selected to generate the cut, the pivot row is

$$x' = \left[\frac{a_0}{\lambda}\right] + \sum_{j=1}^{n} \left[\frac{a_{1j}}{\lambda}\right] (-x_{J(j)}),$$

where $[a_0/\lambda] < 0$. If a_{1j} $(j=1,\ldots,n)$ is non-negative, the integer program is infeasible. Otherwise there is an index p such that $a_{1p} < 0$. Since λ is chosen so that the pivot $[a_{1p}/\lambda] = -1$ after pivoting, we have a_{00} decreasing, which is a contradiction. (This follows because its new value $a'_{00} = a_{00} + a_{0p}\,[a_{10}/\lambda]$, $a_{0p} > 0$ and $[a_{10}/\lambda] < 0$.) Thus, a_{10} must eventually remain fixed at a non-negative integer. The word-for-word argument can now be repeated for a_{20}. That is, since a_{10} is fixed, a_{20} is bounded below by 0, and therefore it also must eventually remain constant. The argument is repeated for all indices i ($i = 1,\ldots, m+n$).

Observe that the proof supposed that the first eligible source row will eventually be selected to generate the cut. Hence, any source row selection rule that has this property will support a finite algorithm.

2.4 RELATION OF FRACTIONAL CUT TO THE ALL INTEGER CUT

The all integer cut is

$$x' = \left[\frac{a_0}{\lambda}\right] + \sum_{j=1}^{n} \left[\frac{a_j}{\lambda}\right](-x_{J(j)}) + \left[\frac{1}{\lambda}\right](-x) \qquad \ldots(2.1)$$

where the source row is

$$x = a_0 + \sum_{j=1}^{n} a_j (-x_{J(j)}) ; \qquad \ldots(2.2)$$

X' is a non-negative integer variable. Then the all integer cut was obtained by selecting $\lambda > 1$. Since (2.1) is true for any postive λ, we may take $\lambda = 1$. Then, $1/\lambda = 1$, and using (2.2), equality (2.1) becomes

$$x' = [a_0] + \sum_{j=1}^{n} [a_j](-x_{J(j)}) - \left\{a_0 + \sum_{j=1}^{n} a_j(-x_{J(j)})\right\},$$

or $$x' = -f_0 + \sum_{j=1}^{n} (-f_j)(-x_{J(j)}).$$

The last equality is Gomory fractional cut.

2.5 PRIMAL ALL INTEGER PROGRAMMING ALGORITHMS

Much of the research on integer programming algorithms has concentrated on dual algorithms. The importance of primal algorithm has been stressed, but until recently only very rudimentary primal algorithms were available. The procedure

requires an all integer primal feasible initial tableau. It adds Gomory cut at each iteration, starting with the very first, so as to maintain an all integer tableau and primal feasibility. When dual feasibility is reached the tableau is integer optimal and the integer program is solved. As the algorithm uses Gomory all integer cuts it can not solve a mixed integer program.

The " Direct Algorithm ", a primal technique, was first described by Ben-Israel and Charnes [1962]. That algorithm, however, frequently required the solution of a sometimes difficult integer programming " auxiliary problem ". Richard D.Young [1965] proposed a different primal algorithm, labeled the 'Rudimentary Primal Algorithm", which avoided the auxiliary problem. However, the technique was somewhat complicated and difficult to implement. Fred Glover [1967] proposed a pseudo primal-dual algorithm which utilizes Gomory's dual all integer method [1963] with a variation of Young's primal all integer technique [1965]. Subsequently, Glover [1968] and Young [1968] drawing from their and each others work, developed simplified primal integer programming algorithms. The contents of these papers are

essentially built on the same foundations and, in fact often overlap. There are, however, some differences between the methods. For one thing, they do not employ the same algorithm strategies, tableau format, and notation. A more important difference is in the use of a " reference equation " introduced to find the pivot column and in the rules which select the source row from which the Gomory cut or pivot row is generated. Although both papers present the general necessary properties of the reference row and the source row selection rule, Young, for ease of exposition and implementation, outlines an algorithm about a particular reference row and source row selection criterion. Glover, on the other hand, does not specialise these, thus creating a more general approach. The simplified algorithm, outlined by Young is then treated as a special case.

2.5.1 The Rudimentary Primal Algorithm :

Step I- Start with a primal feasible all integer tableau. If such a tableau can not be found the integer program is infeasible- terminate. Go to step II.

Step II- Find the pivot column indexed by p using

$$a_{op} = \underset{a_{oj} < o}{\text{minimum}} \; a_{oj} \qquad (j \geq 1)$$

If $a_{oj} \geq o$ $(j = 1,\ldots, n)$, the tableau is integer optimal — terminate. Go to step III.

Step III- Find the row indexed by v utilizing

$$\theta_p = \frac{a_{vo}}{a_{vp}} = \underset{a_{ip} > o}{\text{minimum}} \left\{\frac{a_{io}}{a_{ip}}\right\}$$

Arbitrarily break ties. If $a_{ip} \leq o$ $(i = 1,\ldots, n+m)$, the integer program has an unbounded solution — terminate. Go to step IV.

Step IV- From a row i with $a_{ip} > o$ that satisfies $[a_{io}/a_{ip}] \leq \theta_p$, generate a Gomory all integer cut. Set the parameter λ in the cut equal to a_{ip}, let the derived row be the pivot row. Perform a primal simplex pivot step (the pivot element is $[a_{ip}/a_{ip}]=1$) and return to step II.

2.5.2 Numerical Example :

$$\begin{aligned} \text{Maximize} \quad & x_1 + x_2 = x_o \\ \text{Subject to} \quad & 2x_1 \leq 7 \\ & -x_1 + x_2 \leq o \\ & x_1 , x_2 \geq o, \text{ integer} \end{aligned}$$

With non-negative slack variables x_3 and x_4 we have tableau 1. As $-1 = -1$, there is tie. Now arbitrarily choosing column 2 is the pivot column. The x_4 row generates the cut since minimum $\{7/0, 0/1\} = 0$. The Gomory slack is denoted by S_1. Its row is used as pivot row, and after a pivot we obtain tableau 2. The computations are then repeated. It requires one more cut S_2 to finish the problem. It appears in tableaux 2 and 3. The circled entry corresponds to the pivot element.

Tableau-1

	1	$-x_1$	$-x_2$
x_0	0	-1	-1
x_1	0	-1	0
x_2	0	0	-1
x_3	7	2	0
x_4	0	-1	1
source row			
G.Cut S_1	0	-1	(1)

$(x_1 = x_2 = 0)$

$p = 2$

$\lambda = 1$

Tableau-2

	1	$-x_1$	$-s_1$
x_0	o	-2	1
x_1	o	-1	o
x_2	o	-1	1
→ x_3	7	2	o
source row x_4	o	o	-1
s_2	3	(1)	o

$(x_1 = x_2 = o)$

$p = 1$

$\lambda = 2$

As $-2 < +1$, column 2 is pivot column. x_3 is source row since 7/2 is the only positive ratio.

Tableau-3

	1	$-s_2$	$-s_1$
x_0	6	2	1
x_1	3	1	o
x_2	3	1	1
x_3	1	-2	o
x_4	o	o	-1

$(x_1 = x_2 = 3, x_0 = 6)$

Primal All Integer Solution

Writing the added inequalities in terms of x_1 and x_2, the original non-basic variables, yields :

$$S_1 = o + x_1 - x_2 \geq o \quad \text{or} \quad x_1 - x_2 \geq o$$

$$S_2 = 3 - x_1 - o.S_1 \geq o \quad \text{or} \quad 3 - x_1 \geq o \text{ or } x_1 \leq 3$$

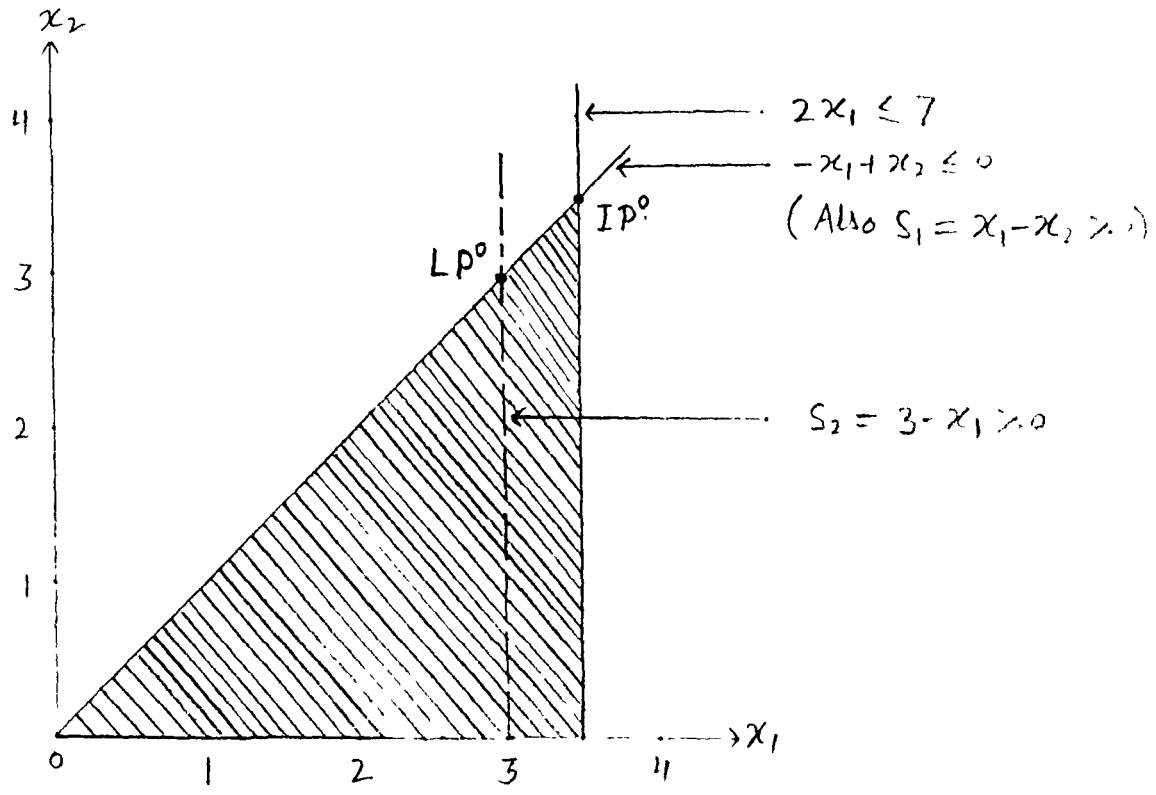


Figure 2.4- A graph of the Numerical Example 2.5.2 showing solutions, constraints, and cut constraints.

Observe that tableaux 1 and 2 correspond to the same point. This happened because the constant element in the Gomory cut was o in tableau 1. Thus, after pivoting, the zero column did not change. If fact, this may occur for an infinite number

of tableaux, in which case the RPA will not converge (see Mathis [1971]). To eleminate the possibility of this phenomenon RPA is modified and simplified Primal algorithms are developed that also guarantee a finite algorithm.

2.5.3 Young's Simplified Primal Algorithm :

The convergent algorithm developed, entitled the Simplified Primal Algorithm (SPA), is essentially the Rudimentary Algorithm with three modifications. The first is the adjoining of a primal feasible all integer equation, indexed by L, whose coefficients $a_{Lj}(j = 0,1,\ldots,n)$ satisfy certain properties with relation to the tableau's columns. The second is the selection of the pivot column, which is accomplished by utilizing the coefficients a_{Lj}. The third modification is concerned with the selection of the source row. This is usually done by reference to the size of the coefficients in the pivot column.

Initially Young adjoins to the bottom of the tableau an all integer constraint or " reference row " which is satisfied by every integer feasible solution. This row indexed by L , is written as

$$x_L = a_{Lo} + \sum_{j=1}^{n} a_{Lj}(-x_j) \geq o \qquad \ldots(2.3)$$

where x_L is a non-negative slack variable. To agree with the pivot column selection rule and to guarantee finiteness, we require that the coefficients in reference row satisfy

(I) $\alpha_j \overset{f}{<} o \implies a_{Lj} > o \quad (j=1,\ldots,n)$

and

(II) $a_{Lj} < o \implies r_j \overset{f}{<} r_p \quad (j=1,\ldots,n)$

Requirement (I) ensures that each column indicating that a dual variable has a non-positive value will be considered as a candidate for the pivot column. In particular, if the tableau is not optimal the pivot column always exists. The first and second requirements with $a_{Lp} > o$ guarantees that

$$a_{Lj}\alpha_p \overset{f}{<} a_{Lp}\,\alpha_j \quad \text{for all} \quad j \geq 1.$$

For each column $\alpha_j (j \geq 1)$ in the tableau having $a_{Lj} \neq o$, define the vector $r_j = \alpha_j / a_{Lj}$. The pivot column α_p is then selected so that

$$r_p \overset{f}{<} r_j \quad \text{for every } j \text{ with } a_{Lj} > o \qquad \ldots(2.4)$$

Observe that, since the minus identity matrix is always present

in the tableau, there can not exist two columns α_s, α_k with $s \neq k$ $(s,k \geq 1)$ such that $r_s = r_k$. Hence the inequality (2.4) gives a unique pivot column determination. Also, note that if there exists an $a_{Lj} > 0$ $(j \geq 1)$ with $\alpha_j \lesseqgtr 0$, then r_p exists and is lexicographically negative.

Recall from the RPA that once the pivot column α_p has been found we may determine

$$\theta_p = \underset{a_{ip}>0}{\text{minimum}} \left(\frac{a_{io}}{a_{ip}} \right)$$

then, to maintain primal feasibility, only constraints which satisfy $[a_{io}/a_{ip}] \leq \theta_p$ with $a_{ip} > 0$ $(i \geq 1)$ can be selected as the generating row. Let $V(p)$ be the set of these "legitimate source rows " specifically,

$$V(p) = \left\{ i/_o \leq \left[\frac{a_{io}}{a_{ip}} \right] \leq \theta_p,\ a_{ip} > 0,\ i=1,\ldots,m+n,L \right\}$$

If there is only one element in $V(p)$ there is no choice as to the generating row. However, when more than one such row is available we must, to ensure convergence, invoke an "acceptable source row selection rule ". The following are acceptable rules.

Rule 1- Choose any row in V(p) that ensures $a_{Lp} \leq a_{Lo}$ at finite intervals (i.e. after a finite number of tableaux), and which also periodically reduces a_{ip}/a_{Lp} for the smallest $i \geq 1$ such that $(a_{ip}/a_{Lp}) > a_{io}$.

Let the source row be indexed by i. Thus the pivot row is

$$s = \left[\frac{a_{io}}{a_{ip}}\right] + \sum_{j=1}^{n} \left[\frac{a_{ij}}{a_{ip}}\right] (-x_{J(j)}).$$

(J is the set of indices corresponding to the current nonbasic variables and J(j) is the jth element in J). Then the subsequent to the pivot ,

$$\bar{a}_{ip} = -a_{ip} \quad \text{and} \quad \bar{a}_{ij} < a_{ip} \quad \text{for} \quad j \neq p$$

(Bars denote elements after pivoting)

Rule 2- At finite intervals designate the row from V(p) that will result in $\bar{a}_{op} > a_{op}$ as the source row.(Bars denote terms in the next tableau.) Continue to use this rule until there are none that satisfy it. When this occurs select any row from V(p) as the generating row.

2.5.4 The Simplified Primal Algorithm (SPA) :

Step 1- Start with a primal feasible all integer tableau. If such a tableau can not be found, the integer program is infeasible terminate. Define a reference row, indexed by L, which does not eleminate any integer feasible solutions and also has the properties (I) $\alpha_j \overset{f}{<} o$ implies $a_{Lj} > o$ and (II) $a_{Lj} < o$ implies $r_j \overset{f}{<} r_p$ (p is the pivot column index). Adjoin this row to the bottom of the tableau. Go to step 2.

Step 2- For each j with $a_{Lj} \neq o$ define $r_j = \alpha_j / a_{Lj}$. Select the pivot column indexed by p so that $r_p \overset{f}{<} r_j$ for every $j \geq 1$ with $a_{Lj} > o$. If $r_p \overset{f}{\geq} o$ or it can not be defined, the tableau is optimal — terminate. Go to step 3.

Step 3- Define the set of legitimate source row

$$V(p) = \left\{ i/o \leq \left[\frac{a_{io}}{a_{ip}} \right] \leq \theta_p,\ a_{ip} > o,\ i=1,\ldots,m+n, i \right\},$$

where $\theta_p = \underset{a_{ip} > o}{\text{minimum}} (a_{io}/a_{ip})$. Select a source row v from $V(p)$ according to an acceptable source row selection rule. Generate from this row an all integer Gomory cut where $\lambda = a_{vp}$. Use this constraint as the pivot row. Go to step 4.

Step 4- Perform a primal simplex pivot operation and return to step 2.

2.5.5 Numerical Example :

$$\begin{aligned} \text{Maximise} \quad & x_1 + x_2 + x_3 = x_0 \\ \text{Subject to} \quad & -4x_1 + 5x_2 + 2x_3 \leq 4 \\ & -2x_1 + 5x_2 \leq 5 \\ & 3x_1 - 2x_2 + 2x_3 \leq 6 \\ & 2x_1 - 5x_2 \leq 1 \\ & x_1 , x_2 , x_3 \geq 0, \text{ integer} \end{aligned}$$

In this example, we shall use $x_1 + x_2 + x_3 \leq 10$ as the L constraint The number 10 was selected by an inspection of the constraints. Also, the source row will be selected from V(p) through longest sequence of preceeding tableaux. Ties are broken arbitrarily. With non-negative slack variables x_4, x_5, x_6 and x_7 we have the first tableau. It requires a total of seven iterations to reach optimality.

Tableau-1

		1	$-x_1$	$-x_2$	$-x_3$
	x_0	0	-1	-1	-1
	x_4	4	-4	5	2
	x_5	5	-2	5	0
	x_6	6	3	-2	2
→ source row	x_7	1	2	-5	0
	x_L	10	1	1	1
	x_1	0	-1	0	0
	x_2	0	0	-1	0
	x_3	0	0	0	-1
→ Cut	s_1	0	①	-5	0

$p = 1$, $\theta_p = \frac{1}{2}$

$V(p) = 4$

$\lambda = 2$

Tableau-2

		1	$-s_1$	$-x_2$	$-x_3$
	x_0	0	1	-4	-1
	x_4	4	4	-7	2
	x_5	5	2	-1	0
→ source row	x_6	6	-3	7	2
	x_7	1	-2	1	0
	x_L	10	-1	4	1
	x_1	0	1	-3	0
	x_2	0	0	-1	0
	x_3	0	0	0	-1
Cut →	s_2	0	-1	①	0

$p = 2$, $\theta_p = 6/7$

$V(p) = 3$

$\lambda = 7$

Tableau-3

		1	$-s_1$	$-s_2$	$-x_3$
	x_0	0	-3	4	-1
	x_4	4	-3	7	2
	x_5	5	1	1	0
→	x_6	6	4	-7	2
source row	x_7	1	-1	-1	0
	x_L	10	3	-4	1
	x_1	0	-2	3	0
	x_2	0	-1	1	0
	x_3	0	0	0	-1
Cut→	s_3	1	(1)	-2	0

$p = 1$, $\theta_p = 3/2$

$V(p) = \{3\}$

$\lambda = 4$

Tableau-4

		1	$-s_3$	$-s_2$	$-x_3$
	x_0	3	3	-2	-1
	x_4	7	3	1	2
	x_5	4	-1	3	0
→	x_6	2	-4	1	2
source row	x_7	2	1	-3	0
	x_L	7	-3	2	1
	x_1	2	2	-1	0
	x_2	1	1	-1	0
	x_3	0	0	0	-1
Cut→	s_4	1	-1	(1)	0

$P = 2$, $\theta_p = 4/3$

$V(p) = \{2\}$

$\lambda = 3$

Tableau-5

	1	$-S_3$	$-S_4$	$-x_5$
x_0	5	1	2	-1
x_4	6	4	-1	2
x_5	1	2	-3	o
→ x_6 source row	1	-3	-1	2
x_7	5	-2	3	o
x_L	5	-1	-2	1
x_1	3	1	1	o
x_2	2	o	1	o
x_3	o	o	o	-1
Cut→ S_6	o	-2	-1	(1)

$p = 3$, $\theta_p = 1/2$

$V(p) = \{3\}$

$\lambda = 2$

Tableau-6

	1	$-S_3$	$-S_4$	$-S_5$
x_0	5	-1	1	1
x_4	6	8	1	-2
→ x_5 source row	1	2	-3	o
x_6	1	1	1	-2
x_7	5	-2	3	o
x_L	5	1	-1	-1
x_1	3	1	1	o
x_2	2	o	1	o
x_3	o	-2	-1	1
Cut → S_6	o	(1)	-2	o

$p = 1$, $\theta_p = 1/2$

$V(p) = \{1,2\}$

$\lambda = 2$

Observe that at tableau 6 we arbitrarily selected second row (i.e. the x_5 equation) as the generating row. Then at tableau 7 row 1 appears in V(p) (and row 2 does not) so it is selected as the source row. Tableau 8 is optimal with $x_0=5$, $x_1=3$, $x_2=2$ and $x_3=0$

Tableau-7

		1	$-s_6$	$-s_4$	$-s_5$	
	x_0	5	1	-1	1	
	x_4	6	-8	17	-2	
source row →	x_5	1	-2	1	0	$p = 4$, $\theta_p = 1/3$
	x_6	1	-1	3	-2	$V(p) = \{1,3\}$
	x_7	5	2	-1	0	$\lambda = 17$
	x_8	5	-1	1	-1	
	x_1	3	-1	3	0	
	x_2	2	0	1	0	
	x_3	0	2	-5	1	
Cut →	s_7	0	-1	(1)	-1	

Tableau 8

	1	$-S_6$	$-S_7$	$-S_5$	
x_o	5	o	1	o	$(x_1=3,\ x_2=2,\ x_3=o,\ x_o=5)$
x_4	6	9	-17	15	Integer optimal solution
x_5	1	-1	-1	1	
x_6	1	2	-3	1	
x_7	5	1	1	-1	
x_L	5	o	-1	o	
x_1	3	2	-3	3	
x_2	2	1	-1	1	
x_3	o	-3	5	-4	

The constraints x_L, S_1, S_2, S_3, S_4, S_5, S_6, and S_7 i form of x_1, x_2 and x_3 are :

$x_L = x_1 + x_2 + x_3 \leq 10$

$S_1 = o - x_1 + 3x_2 \geq o$ or $S_1 = 3x_2 - x_1 \geq o$

$S_2 = o + S_1 - x_2 \geq o$ or $S_2 = 3x_2 - x_1 - x_2 \geq o$

or $S_2 = 2x_2 - x_1 \geq o$

$S_3 = 1 + x_2 - x_4 \geq o$

$S_4 = 8 - x_2 \geq o$ or $x_2 \leq 2$

$$S_5 = 4 - 2x_1 + x_2 - x_3 \geq 0$$

$$S_6 = 1 + x_1 - 2x_2 \geq 0$$

$$S_7 = 3 - x_1 - x_3 \geq 0$$

2.5.6 Finiteness : The strategy of finiteness proof is simple under the following rules :

Rule 1- Choose any row in $V(p)$ that ensures $a_{Lp} \leq a_{Lo}$ at finite intervals and also periodically reduces a_{ip}/a_{LP} for the smallest $i \geq 1$ such that $a_{ip}/a_{Lp} > a_{io}$.

Rule 2- At finite intervals designate the row from $V(p)$ that will result in $\bar{a}_{op} > a_{op}$ as the source row. Continue to use this rule until there are none that satisfy it. When this occurs, select any i from $V(p)$ as the generating row.

Rule 3- In addition to the reference row the equation

$$x_q = a_{qo} + \sum_{j=1}^{n} (-a_{oj})(-x_j),$$

where $-a_{qo}$ is a non-positive lower bound for $\sum_{j=1}^{n} a_{oj} x_j$, is added to the initial tableau. Then Rule 3 is the same rule 1 except that $a_{Lp} \leq a_{Lo}$ is replaced by $-a_{op} \leq a_{qo}$ and the algorithm terminates when $a_{Lo} = o$ or $a_{op}/a_{Lp} > -1/a_{Lo}$.

2.5.7 Glover's Primal Algorithm :

Fred Glover also uses an auxiliary constraint in his algorithm. After observing that Young's L-row is a legitimate auxiliary constraint, he proves that any feasible solution to the dual of the original linear programming problem can be used as multipliers of the constraints to form an auxiliary constraint. Thus Glover's auxiliary constraint is a surrogate constra int (a positive linear combination of constraints which yields a constraint that is " strong " in some, to be defined, sense). He proposes using the optimal solution to the dual as multipliers of the constraints , but does not imply that this is necessarily superior to using any feasible solution. He does not propose changing the constraint periodically, as does Young, but indicates the possibility ofdoing so.

The selection of an incoming variable in Glover's algorithm is exactly the same as in Young's, with the auxiliary const. nt in place of L-row, except that Glover does not seek out transition cycles.

Selection of the constraint to be used for generating a cut

constraint is the same as that of Young except that the previous iteration is not considered. Glover breaks any ties by choosing the iteration which results in the maximum sum of the negative a_{oj} in the next tableau. In any resulting ties the iteration with smallest index is chosen. Although the latter procedure is heuristic in nature, it does ensure convergence, proof of which is given in Glover's paper [1968].

Let $\bar{w}$ be a feasible solution to the dual linear program. Then the constraint

$$x_L = a_{Lo} + \sum_{j=1}^{n} a_{Lj} (-x_j) \geq 0$$

where $a_{Lo} = \bar{w}b$ and $a_{Lj} = \bar{w}_{aj}$, may serve as the reference row. That is, it has all the required properties.

2.5.8 Numerical Example :

$$\begin{aligned} \text{Maximise} \quad & 4x_1 + 6x_2 + 3x_3 = x_0 \\ \text{Subject to} \quad & x_1 + 2x_2 \leq 5 \\ & 9x_1 + 2x_2 - 4x_3 \leq 8 \\ & -3x_1 - 2x_2 + 2x_3 \leq 1 \\ & -5x_1 + 4x_2 + 6x_3 \leq 16 \\ & x_1, x_2, x_3 \geq 0, \text{ integer} \end{aligned}$$

Before giving the first tableau we ought to mention that the reference row defined by the dual variables will be employed. However, to maintain all integer tableaux the constraint will initially be cleared of fractions. So to obtain the L-row we omit the x_j integer ($j = 1,2,3$) requirements from the program and then pass to its dual. This problem is

$$\begin{aligned} \text{Minimize} \quad & 5w_1 + 8w_2 + w_3 + 16w_4 \\ \text{Subject to} \quad & w_1 + 9w_2 - 3w_3 - 5w_4 \geq 4 \\ & 2w_1 + 2w_2 - 2w_3 + 4w_4 \geq 6 \\ & -4w_2 + 2w_3 + 6w_4 \geq 3 \\ & w_1 , \ w_2 , \ w_3 , \ w_4 \geq 0 \end{aligned}$$

We choose the optimal dual feasible solution to determine the coefficients of the reference row. After solving the dual problem we get $w_1 = 0$, $w_2 = 39/34$, $w_3 = 0$ and $w_4 = 43/34$. Then

$$a_{Lo} = (0, 39/34, 0, 43/34) \begin{pmatrix} 5 \\ 8 \\ 1 \\ 16 \end{pmatrix} = 1000/34$$

and

$$(a_{L1}, a_{L2}, a_{L3}) = (0, 39/34, 0, 43/34) \begin{pmatrix} 1 & 2 & 0 \\ 9 & 2 & -4 \\ -3 & -2 & 2 \\ -5 & 4 & 6 \end{pmatrix} = 1/34(136, 250, 102)$$

Hence, the L constraint is

$$(136/34)x_1 + (250/34)x_2 + (102/34)x_3 \leq 1000/34$$

or, after multiplying by 17 and adding the non-negative slack variable x_L, the reference row is

$$x_L = 500 + 68(-x_1) + 125(-x_2) + 51(-x_3) \geq 0$$

Adjoining this row to the constraint set and adding the non-negative slack variables x_4, x_5, x_6, and x_7 to each of the four original inequalities yields tableau 1

Tableau-1

		1	$-x_1$	$-x_2$	$-x_3$
	x_0	0	-4	-6	-5
	x_4	5	1	2	0
	x_5	8	9	2	-4
source→	x_6	1	-3	-2	2
row	x_7	16	-5	4	6
	x_L	500	68	125	51
	x_1	0	-1	0	0
	x_2	0	0	-1	0
	x_3	0	0	0	-1
Cut →	s_1	0	-2	-1	(1)

$p = 3$, $\theta_p = 1/2$

$V(p) = \{3\}$

$\lambda = 2$

Note that a_{01}, a_{02}, and a_{03} are negative. Hence, it is unnecessary to consider the coefficients in x_4 row, or to insert a new constraint between x_0 and x_4 to support the lexicographic ordering. Finally, we shall select the source row according to a slight variation of Glover's Rule 2. Specifically, whenever possible, the legitimate source row which maximizes $\bar{a}_{0p}$ is picked. Ties are broken by selecting the row which maximizes the sum of the negative $\bar{a}_{0j}$ ($j \geq 1$) and then by selecting the smallest index. The computations appear in the seven tableaux.

Tableau-2

		1	$-x_1$	$-x_2$	$-s_1$	
	x_0	0	-10	-9	3	
	x_4	5	-1	2	0	$p = 1$, $\theta_p = 1$
	x_5	8	1	-2	4	
Cut and source row	x_6	1	(1)	0	-2	$V(p) = \{3\}$
	x_7	16	7	10	-6	$\lambda = 1$
	x_L	500	170	176	-51	
	x_1	0	-1	0	0	
	x_2	0	0	-1	0	
	x_3	0	-2	-1	1	

Since $\lambda = 1$, a cut is not added to the tableau 2. In the next tableau the e are three legitimate source rows : namely rows 2,4 and 5. If row 2 is used then $\lambda = 6$, $\bar{p} = 1$, and $\bar{a}_{0\bar{p}} = -7$, when row 4 is picked, $\lambda = 8$, $\bar{p} = 1$, and $\bar{a}_{0\bar{p}} = -7$, finally row 5 yields $\lambda = 289$, $\bar{p} = 2$, and $\bar{a}_{0\bar{p}} = -9$. A tie exists between rows 2 and 4. Row 2 would result in $\bar{a}_{01} = -7$, $\bar{a}_{02} = -26$, and $\bar{a}_{03} = 17$, and if row 4 is chosen, we get $\bar{a}_{01} = -7$, $\bar{a}_{02} = 8$, and $\bar{a}_{03} = 17$. As $-7 > -7\ -26$, row 4 is selected. Similar computations are performed at tableaux 4, 5 and 6.

Tableau-3

		1	$-x_6$	$-x_2$	$-s_1$	
	x_0	10	10	-9	-17	
	x_4	4	-1	2	2	
	x_5	7	-1	-2	6	$p = 3,\ \theta_p = 9/8$
	x_6	0	-1	0	0	
	x_7	9	-7	10	8	$V(p) = \{2,4,5\}$
source row →	x_L	330	-170	176	289	$\lambda = 8$
	x_1	1	1	0	-2	
	x_2	0	0	-1	0	
	x_3	2	2	-1	-3	
Cut →	s_2	1	-1	1	(1)	

Tableau-4

	1	$-x_6$	$-x_2$	$-s_2$
x_0	27	-7	8	17
x_4	2	1	0	-2
→ x_5 source row	1	5	-8	-6
x_6	0	-1	0	0
x_7	1	1	2	-8
x_L	41	119	-113	-289
x_1	3	-1	2	2
x_2	0	0	-1	0
x_3	5	-1	2	3
Cut→ s_3	0	(1)	-2	-2

$p = 1$, $\theta_p = 1/5$

$V(p) = \{2,5\}$

$\lambda = 5$

The primal feasible solution exhibited in tableau 4 is optimal. Nevertheless, three more iterations are needed to achieve dual feasibility

Tableau-5

		1	$-s_3$	$-x_2$	$-s_2$
	x_0	27	7	-6	3
	x_4	2	-1	2	0
	x_5	1	-5	2	4
	x_6	0	1	-2	-2
	x_7	1	-1	4	-6
source row →	x_L	41	-119	125	-51
	x_1	3	1	0	0
	x_2	0	0	-1	0
	x_3	5	1	0	1
Cut →	s_4	0	-1	(1)	-1

$p = 2$, $\theta_p = 41/125$

$V(p) = \{2,4,5\}$

$\lambda = 125$

Tableau-6

		1	$-S_3$	$-S_4$	$-S_2$	
	x_0	27	1	6	-3	
	x_4	2	1	-2	2	
	x_5	1	-3	-2	6	
	x_6	0	-1	2	-4	$p = 3$, $\theta_p = 1/6$
	x_7	1	3	-4	-2	$V(p) = \{2, 5\}$
source row →	x_L	41	6	-125	74	$\lambda = 74$
	x_1	3	1	0	0	
	x_2	0	-1	1	-1	
	x_3	5	1	0	1	
Cut →	S_5	0	0	-2	(1)	

Tableau-7

		1	$-S_3$	$-S_4$	$-S_5$	
	x_0	27	1	0	3	
	x_4	2	1	2	-2	Integr optimal
	x_5	1	-3	10	-6	$x_0 = 27$
	x_6	0	-1	-6	4	$x_1=3$, $x_2=0$, $x_3=5$
	x_7	1	3	-8	2	
	x_L	41	6	23	-74	
	x_1	3	1	0	0	
	x_2	0	-1	-1	1	
	x_3	5	1	+2	-1	

Tableau 7 exhibits the optimal integer solution. It is worth mentioning that, for this problem, the reference row reduced the number of iterations. In particular, Glover [1968] points out that had $x_L = 8 + (-x_1) + (-x_2) + (-x_3)$ been used with same source row selection rules it would have taken 15 iterations to obtain the optimal tableau.

—

CHAPTER-III

BRANCH AND BOUND METHOD

3.1 INTRODUCTION

The branch-and-bound method generates a sequence of subproblems that differ from one another only in the bounds on variables. The solutions of the subproblems are used to deduce that certain subsets of integer points could not be optimal for the original integer program, and these subsets are systematically eliminated from further consideration. The method is sure eventually to generate a subproblem whose solution can be deduced to be optimal for the original integer program. The method is named branch and bound because it proceeds by branching and then reasoning, on the basis of objective function bounds, that some subset of the possible integer solutions can not contain the optimal point. The process of deciding that a subset of the possible solutions can be excluded from further consideration is called fathoming. Fathoming produces the same result as would be obtained by explicitly enumerating all of the possible solutions in the

subset of discovering that none of them is optimal, but it does so without actually examining all of the points. For this reason the branch-and-bound method is called an implicit enumerating scheme.

An exposition of the classical Land and Doig enumeration algorithm (1960) and of the variations which have since appeared in the literature is given. The method described in Land and Doig [1960] was specialized for the traveling salesman problem by Little, Murty, Sweeney, and Karel [1963]. They termed the specialized method as branch and bound, which, as mentioned in Balinski [1965], is also an apt designation for the Land and Doig algorithm. Thompson [1964] presented an algorithm for the integer program," The Stopped Simplex Method", which is a consequence of Land and Doig's work. A year later, Dakin [1965] proposed a simple, yet interesting, variation of the Land and Doig algorithm. Beale and Small [1965] extended the Dakin [1965] method to include the linear programming post optimization procedures suggested by Driebeek [1966]. Later, Tomlin [1970], [1971], described a refined version of the Beale and Small [1965] algorithm. The Balinski [1965], Balinski and Spielberg [1969]

Geoffrion and Marsten [1972], and Lawler and Wood [1966] papers are survey articles which, among other things, contain an exposition of the Land and Doig [1960] method. A general description of branch and bound may be found in Agin [1966] and, in Balas [1968] or Mitten [1970]. A graph theoretic interpretation appears in Bertier and Roy [1967]. The underlying properties which characterize an algorithm to be of the branch and bound type appeared in Spielberg [1968], and subsequently in Comlin and Spielberg [1969], and Balinski and Spielberg [1969]. A procedure which specializes the Land and Doig approach to the zero one case and incorporates the work of Driebeek [1966] and also the algorithm branching strategies proposed by Spielberg [1968] is described in Davis, Kendrick, and Weitzman [1971].

Recently, Cooper, Mary W. [1981] has classified and discussed algorithms for solution of nonlinear pure integer programming problems. This survey is organized by characterising the mathematical form of the nonlinear optimization problems addressed by the various algorithms. In surveying the papers the dominant ideas in use are those of dynamic methodology

and branch and bound methods. Gupta and A.Ravindran [1983] have also presented a survey of nonlinear integer programming algorithms. Gupta, Omprakash K., Ravindran A. [1985] have investigated the feasibility of applying the branch and bound approach to nonlinear convex integer programming problems. hey have tested the concepts of pseudo-costs and estimation in selecting branching variables and branching nodes. It is to be noted that even though the branch and bound method is not guaranteed to solve nonconvex problems, yet Gupta, Omprakash K, Ravindran A [1985] have successfully solved several nonconvex problems using the BBNLIP code.

In this chapter, we will present the mixed integer program and two classical branch and bound methods. Some partitioning algorithms will also be discussed.

5.2 THE PROBLEM :

The problem under consideration is mixed integer program (MIP)

$$\text{Maximize} \quad z$$

$$\left.\begin{array}{lll} \text{Subject to} & cx + dy - z = 0 & \ldots(1) \\ & Ax + Dy \leq b & \ldots(2) \\ & x \geq 0, \; y \geq 0 & \ldots(3) \\ & x = (x_j) \text{ integer} & \ldots(4) \end{array}\right\} \; S$$

where A is m by n, D is m by n', and all other terms are of appropriate size.

The vectors (x,y,z) satisfying (1), (2), and (3) define the set S. The linear program : Maximize z subject to (x,y,z) in S, will be denoted by LP^0. A point or vector x with $(0 \leq l \leq n)$ of its components fixed at non-negative integer values is labeled x^l. The point x^l has n-l not-fixed or free components. The linear program LP^0 with the additional constraint that the l fixed components in x^l are fixed at their integer values, is also a linear program in the free x and the y variables, and will be referred to as LP^l, or more simply, as the subproblem at x^l. The feasible region corresponding to the linear program LP^l is denoted by S^l; equivalently, S^l is the set of points (x^l, y, z) satisfying (1),(2), and (3). Notice that S^l is an $(n-l)+n'+1$ dimensional set in

n+n'+1 space, and also that $S^0 = S$. The optimal solution to the (linear programming) subproblem at x^{l+1}, where x^{l+1} is defined from x^l by setting one of its free variables x_k to the nonnegative integer t, is denoted by $s(l,k,t)$. This value includes the costs of the $l+1$ fixed variables in x^{l+1}. Observe that $s(l,k,t)$ is the optimal solution to the linear program LP^l with the additional constraint $x_k=t$. Finally, the number s^l ($l=0,1,\ldots,n-1$) will be an upper bound for any mixed integer programming solution found from the point x^l.

3.3 THE TREE ALGORITHM FORMULATION :

Enumerative algorithms are usually easier to understand if they are pictorially related to a tree composed of nodes and branches. A node corresponds to a point x^l and a branch links the node x^l with x^{l+1}, where the latter point is defined from the previous one by setting one of its $n-l$ free variables to a nonnegative integer. As a free variable x_k can usually take on several values, it is possible to have many branches emanating from the node x^l. With regard to figure 3.1 the nodes numbered 8,9 and 10 are created by setting x_1, a free variable in node 5, to 3,4 and 2, respectively. Nodes such as these tree which

have not as yet been used to create other nodes, or equivalently, which lack emanating branches are called dangling. For example node 5 is dangling. Also, the level number ℓ refers to the number of fixed variables.

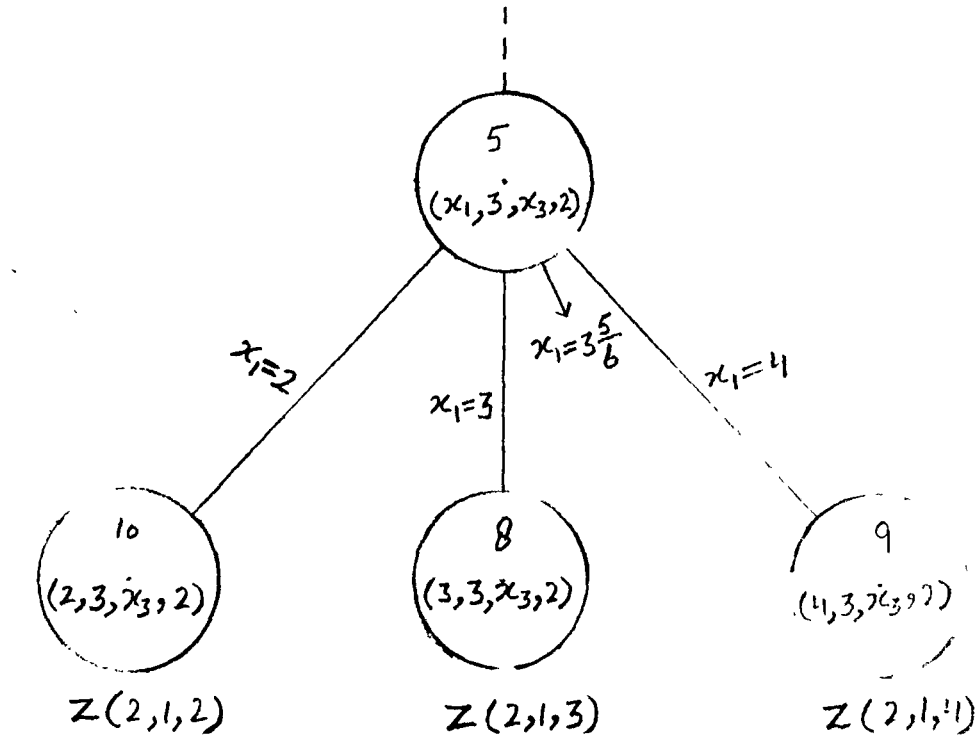


Figure 3.1 Branch abd Bound Tree

Suppose that $x_1 = 3\frac{5}{6}$ in the optimal solution to the subproblem at node 5. Then by fixing x_1 at the next lowest and highest integer, node 8 and 9 are created. Solving the subproblem or linear program at each node yields $z(2,1,3)$ and $z(2,1,4)$. The larger of these two numbers, say $z(2,1,3)$ is an upper bound

z^2 for any mixed integer programming solution found from node 5. By fixing x_1 at 2, node 10 (which is to the immediate left of node 8) is created and $z(2,1,2)$ can be determined. Note that if node 8 is ever eliminated from consideration, z^2 may be replaced by the maximum of $z(2,1,2)$ and $z(2,1,4)$. If the current best mixed integer solution z^* is at least as large as $z(2,1,3)$, then all the nodes which can be traced back to node 5 can not be candidates for improved solutions to MIP. Similarly, if $z^* \geq z(2,1,2)$ ($z^* \geq z(2,1,4)$), then all nodes to the left(right) of node 8 may be implicitly enumerated. So if the current best solution is at least as large as both $z(2,1,2)$ and $z(2,1,4)$, and $z^* < z(2,1,3)$, node 8 alone has to be examined at level 3. Thus we may say that only those nodes at which the optimal linear programming solution exceeds z^*, the current best solution for the MIP, should be eligible to create new nodes. In other words, these are the only ones that should be explicitly examined or remain on the list of dangling nodes. When a linear program at a node is infeasible, then all subproblems at nodes either to the left or right of it are also infeasible.

3.4 LE BASIC APPROACH (Land and Doig [1960])

In the above section the working of algorithm is discussed.

For completeness, a general outline of the Land and Doig procedure is given. It is assumed that each integer variable is bounded above.

3.4.1 Land and Doig Procedure :

Step 1-(Initialization) Set z^*, the current best solution to the MIP, to some arbitrarily small or predetermined value. The initial node, with all variables free, is $x^0 = (x_1, \ldots, x_n)$. Solve LP^0. If it is infeasible, so is the MIP-terminate. If its optimal solution is integer in the integer constrained variables, the MIP is solved - terminate. Otherwise, set $x^l = x^0$ and go to step 2.

Step 2-(Branching) From the optimal linear programming solution $(\hat{x}, \hat{y}, \hat{s})$ at x^l pick a variable $\hat{x}_k$ which does not satisfy the integer requirement. By fixing $\hat{x}_k$ at $|\hat{x}_k|$ and $|\hat{x}_k|+1$, define two nodes from x^l. Solve the subproblem at each of these nodes. Label dangling those created nodes whose optimal linear programming solution exceeds z^*. Check each node for an improved solution to the MIP. If one is found, it is recorded, and all dangling nodes with a subproblem solution not exceeding it are dropped from the list. Go to step 3.

Step 3-(Termination Test) If the current list of dangling nodes is empty, either the optimal solution to the MIP, z^* has been found or no solution exists - terminate. Otherwise, go to step 4.

Step 4-(Bounding) Determine the dangling node x^l which has the largest optimal solution. Break ties arbitrarily. Suppose the point x^{l-1} defines x^l, the selected node, by setting $x_k = t$. (This means the optimal solution to LP^l is $z(l-1,k,t)$.) Set z^{l-1}, an upper bound for any mixed integer programming solution found from x^{l-1}, equal to the optimal linear programming solution $z(l-1,k,t)$ at x^l. Create a node to the immediate left or right of x^l so that if another dangling node created from x^{l-1} is eventually selected, a new (not higher) value for z^{l-1} may be found. Remove x^l from the list of dangling nodes and go to step 2.

Remarks :

1. If the linear program at the node created by fixing a not integral $\hat{x}_k$ at its next lowest integer value, and that the node created by fixing $\hat{x}_k$ at its next highest integer value are both infeasible, there can not exist a solution to the MIP with x_k integer emanating from the node x^l. In particular, if this

occurs at level o (ℓ=o) the MIP has no feasible solution. Geometrically the upper boundary of S^{ℓ} is as in the following figure

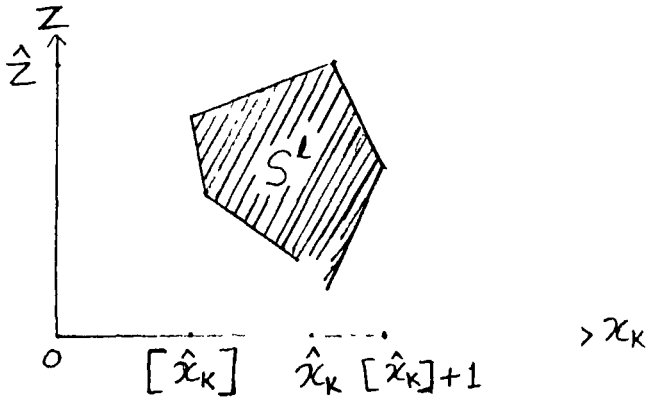


Figure 3.2

2. The upper bound $z^{\ell-1}$, equal to some $z(\ell-1,k,t)$ value, for any mixed integer programming solution found from $x^{\ell-1}$, is replaced (usually by a lower value) when it is ascertained that there can not exist an improved mixed integer solution emanating from x^{ℓ}. Whenever $z^{\ell-1} \leq z^*$, node $x^{\ell-1}$ and all points defined from it can not be candidates for an improved solution to the MIP. Hence, prior to branching from x^{ℓ}, the purpose of creating a level ℓ node to the left or right of it is to have the immediate ability to revise $z^{\ell-1}$, usually downward. This is done by replacing $z^{\ell-1}$ by the maximum of the subproblem solution at the nodes on either side of x^{ℓ}.

3. When a subproblem at level n is feasible, it produces a solution to the MIP. Hence the algorithm converges, since, in

the worst case, all nodes of the tree are enumerated explicitly, and the nodes in the last row represent all possible combinations of integers.

4. To reduce computational effort it is probably worthwhile to explicitly introduce the inequality $cx^{\ell} + dy \geq z^* + \epsilon$, where ϵ is a small positive number, to each linear program L_i^{ℓ}. By doing this, a feasible linear program always indicate a dangling node, and only improved mixed integer solutions can be used.

5. If node x^{ℓ} is defined from $x^{\ell-1}$ by setting x_k, a free variable in $x^{\ell-1}$, to t, it is a simple matter to modify the optimal simplex tableau at node $x^{\ell-1}$ to produce the first tableau at x^{ℓ}.

6. The continual branching and bounding process suggests the branch and bound designation for the algorithm.

3.4.2 Numerical Example : Land and Doig Algorithm

$$\begin{aligned} \text{Maximize} \quad & x_1 + 3x_2 = z \\ \text{Subject to} \quad & 3x_1 + x_2 \leq 5 \\ & 4x_1 + 4x_2 \leq 9 \\ & x_1 , x_2 \geq 0, \text{ integer} \end{aligned}$$

Step 1- The initial node is $x^0 = (x_1, x_2)$. The optimal linear

programming solution with non-negative slack variables x_3 and x_4 is in tableau 1

Tableau-1

	1	$-x_1$	$-x_4$	
z	$6\frac{3}{4}$	2	$\frac{3}{4}$	$z = 6\frac{3}{4}$
x_1	0	-1	0	
x_2	$2\frac{1}{4}$	1	$\frac{1}{4}$	$x_1 = 0,\ x_2 = 2\frac{1}{4}$
x_3	$2\frac{3}{4}$	2	$-\frac{1}{4}$	
x_4	0	0	-1	

Step 2- As x_2 must be integer, it is fixed at $[2\frac{1}{4}]$ and $[2\frac{1}{4}]+1$ to define node 2 or $(x_1,2)$, and node 3 or $(x_1,3)$.

To solve the node 2 linear program we may incorporate the constraint $x_2 = 2$ or equivalently $x_2 \leq 2$ and $x_2 \geq 2$, into tableau 1. As

$$x_2 = 2\frac{1}{4} - x_1 - \frac{1}{4}x_4$$

we have

$$2 - x_2 = -\frac{1}{4} + x_1 + \frac{1}{4}x_4 .$$

Thus defining $\bar{x}_2 = 2-x_2$, and requiring $\bar{x}_2 \geq 0$ yields $x_2 \leq 2$. Similarly

$$-\bar{x}_2 = -2 + x_2 = \frac{1}{4} - x_1 - \frac{1}{4}\ x_4 \geq 0$$

enforces $x_2 \geq 2$. Since $x_2 \not\geq 2$ implies $x_2 \geq 0$, the x_2 row is omitted and tableau 1 becomes

Tableau-2(a)

		1	$-x_1$	$-x_4$
	z	$6\frac{3}{4}$	2	$\frac{3}{4}$
	x_1	0	-1	0
-	$\bar{x}_2$	$-\frac{1}{4}$	(-1)	$-\frac{1}{4}$
	$-\bar{x}_2$	$\frac{1}{4}$	1	$\frac{1}{4}$
	x_3	$2\frac{3}{4}$	2	$-\frac{1}{4}$
	x_4	0	0	-1

Tableau-2b(Node 2)

	1	$-x_2$	$-x_4$
z	$6\frac{1}{4}$	2	$\frac{1}{4}$
x_1	$\frac{1}{4}$	-1	$\frac{1}{4}$
$\bar{x}_2$	0	-1	0
$-\bar{x}_2$	0	1	0
x_3	$2\frac{1}{4}$	2	$-\frac{3}{4}$
x_4	0	0	-1

The tableau 2(a) is dual feasible and prima infeasible. Pivoting on the circled element regains optimality. Noti that it is never really necessary to explicitly carry both the x_2 and $-x_2$ row. This is true because if one row is present, the other one c readily be found if nee ded. It is generally no necessary to explicitly carry linearly dependen rowe. Also note that at the optimal solution both variables x_2 and $-\bar{x}_2$ must have value o.

Similarly, to solve the linear program at node 3, the constraint $x_2 = 3$ or $x_2 \leq 3$ and $x_2 \geq 3$ is added to tableau 1. This is done by replacing the x_2 constraint with the two constraints

$$\bar{\bar{x}}_2 = 3 - x_2 = 3/4 + x_1 + \frac{1}{4} x_4 \geq 0$$

and

$$-\bar{\bar{x}}_2 = -3 + x_2 = -3/4 - x_1 - \frac{1}{4} x_4 \geq 0$$

Or equivalently by adding 1 to the $\bar{\bar{x}}_2$ row and -1 to the $-\bar{\bar{x}}_2$ row in tableau 2(a). In any case we have tableau 3(a). Note that $\bar{\bar{x}}_2 \geq 0$ and $-\bar{\bar{x}}_2 \geq 0$ guarantees that $x_2 = 3$.

These two inequalities show that the solution is infeasible at node 3.

Step 3.4- Thus node 2 is selected and node 4 or $(x_1,1)$ to the left of node 2 is created by $x_2 = [2\frac{1}{4}] - 1 = 1$.

To solve the node 4 linear program we may incorporate the constraint $x_2 = 1$ or equivalently $x_2 \leq 1$ and $x_2 \geq 1$, into tableau 1. As

$$x_2 = 2\frac{1}{4} - x_1 - \frac{1}{4} x_4$$

we have

$$\bar{\bar{x}}_2 = 1 - x_2 = -5/4 + x_1 + \frac{1}{4} x_4 \geq 0$$

and

$$-\bar{\bar{x}}_2 = -1 + x_2 = 5/4 - x_1 - \frac{1}{4} x_4 \geq 0$$

Or equivalently, by subtracting 1 from the $\bar{\bar{x}}_2$ row and -1 from the $-\bar{\bar{x}}_2$ row in tableau 2(a). In any case we have tableau 4(a). Note that $\bar{\bar{x}}_2 \geq 0$ and $-\bar{\bar{x}}_2 \geq 0$ guarantees that $x_2 = 1$.

Tableau-4(a)

	1	$-x_1$	$-x_4$
z	$6\frac{3}{4}$	2	$\frac{3}{4}$
x_1	0	-1	0
- $\bar{\bar{x}}_2$	$-\frac{5}{4}$	(-1)	$-\frac{1}{4}$
$-\bar{\bar{x}}_2$	$\frac{5}{4}$	1	$\frac{1}{4}$
x_3	$2\frac{3}{4}$	2	$-\frac{1}{4}$
x_4	0	0	-1

Tableau-4b(Node 4)

	1	$-\bar{\bar{x}}_2$	$-x_4$
z	$4\frac{1}{4}$	2	1/4
x_1	5/4	-1	1/4
$\bar{\bar{x}}_2$	0	-1	0
$-\bar{\bar{x}}_2$	0	1	0
x_3	1/4	2	-3/4
x_4	0	0	-1

The tableau 4(a) is dual feasible and primal infeasible. Pivoting on the circled element regai optimality.

$z = 4\frac{1}{4}$

$x_1 = 5/4$

$\bar{\bar{x}}_2 = 1 - x_2 = 0 \Rightarrow x_2 = 1$

Since z is maximum at node 2, this node is labeled dangling.

Step 2- The optimal linear programming solution at node 2 had $x_1 = 1/4$ and $x_2 = 2$ (tableau 2b). Thus by setting $x_1 = [1/4] = 0$ and $x_1 = [1/4] + 1 = 1$, node 5 or (0,2) and node 6 or (1,2) are created.

To solve the node 5 linear program we may incorporate the constraint $x_1 = 0$ or equivalently $x_1 \leq 0$ and $x_1 \geq 0$, into tableau 2b. In this case, two new constraints are not necessary since the inequality $x_1 \geq 0$ is already in the tableau. Thus we add the inequality

$$-x_1 = -\frac{1}{4} - \bar{x}_2 + \frac{1}{4} x_4 \geq 0$$

into tableau 2b. The result is tableau 5a.

Tableau-5(a)

	1	$-\bar{x}_2$	$-x_4$
z	$6\frac{1}{4}$	2	1/4
x_1	1/4	-1	1/4
$-x_1$	-1/4	1	(-1/4)
$\bar{x}_2$	0	-1	0
$-\bar{x}_2$	0	1	0
x_3	$2\frac{1}{4}$	2	-3/4
x_4	0	0	-1

Tableau-5(b) (Node 5)

	1	$-\bar{x}_2$	x_1
z	6	3	1
x_1	0	0	1
$-x_1$	0	0	-1
$\bar{x}_2$	0	-1	0
$-\bar{x}_2$	0	1	0
x_3	3	-1	-3
x_4	1	-4	-4

On pivoting circle entry of the tableau 5(a) we get tableau 5(b)

$z = 6$

$x_1 = 0$

$\bar{x}_2 = 2 - x_2 = 0 \Longrightarrow x_2 = 2$

The solution obtained at node 5 (tableau 5b) is the first integer solution and thus considered to be lower bound.

To solve linear program at node 6 we may incorporate the constraint $x_1 = 1$ or equivalently $x_1 \leq 1$ and $x_1 \geq 1$, into tableau 2b. This is done by replacing the x_1 constraint with the two constraints

$$\bar{x}_1 = 1 - x_1 = \frac{3}{4} - \bar{x}_2 + \frac{1}{4} x_4 \geq 0$$

and $$-\bar{x}_1 = -1 + x_1 = -3/4 + \bar{x}_2 - 1/4\, x_4 \geq 0$$

Or equivalently by adding -1 to x_1 row and 1 to $-x_1$ row in

tableau 5(a). In any case, we have tableau 6(a). Note that $\bar{x}_1 \geq 0$ and $-\bar{x}_1 \geq 0$ guarantees that $x_1 = 1$.

Tableau-6(a)

		1	$-\bar{x}_2$	$-x_4$
	s	$6\frac{1}{4}$	2	1/4
	$\bar{x}_1$	3/4	1	-1/4
→	$-\bar{x}_1$	-3/4	(-1)	1/4
	$\bar{x}_2$	0	-1	0
	$-\bar{x}_2$	0	1	0
	x_3	$2\frac{1}{4}$	2	-3/4
	x_4	0	0	-1

Tableau-6(b) (Node 6)

		1	$-\bar{x}_1$	$-x_4$
	s	$4\frac{3}{4}$	2	3/4
	$\bar{x}_1$	0	1	0
	$-\bar{x}_1$	0	-1	0
	$\bar{x}_2$	3/4	-1	-1/4
	$-\bar{x}_2$	-3/4	1	1/4
	x_3	3/4	2	-1/4
	x_4	0	0	-1

After one pivot tableau 6(a) gives tableau 6(b) which exhibits optimality (linear)

$s = 4\frac{3}{4}$

$\bar{x}_1 = 1 - x_1 = 0 \Rightarrow x_1 = 1$

$\bar{x}_2 = 2 - x_2 = 3/4 \Rightarrow x_2 = 5/4$

or $x_2 = 1\frac{1}{4}$

We find that the z is maximum at node 5 and the solution at this node is also integer, thus this solution is optimal and node 4 and node 6 are fathomed.

Hence, the integer optimal solution is

$z = 6, \quad x_1 = 0 , \quad x_2 = 2$

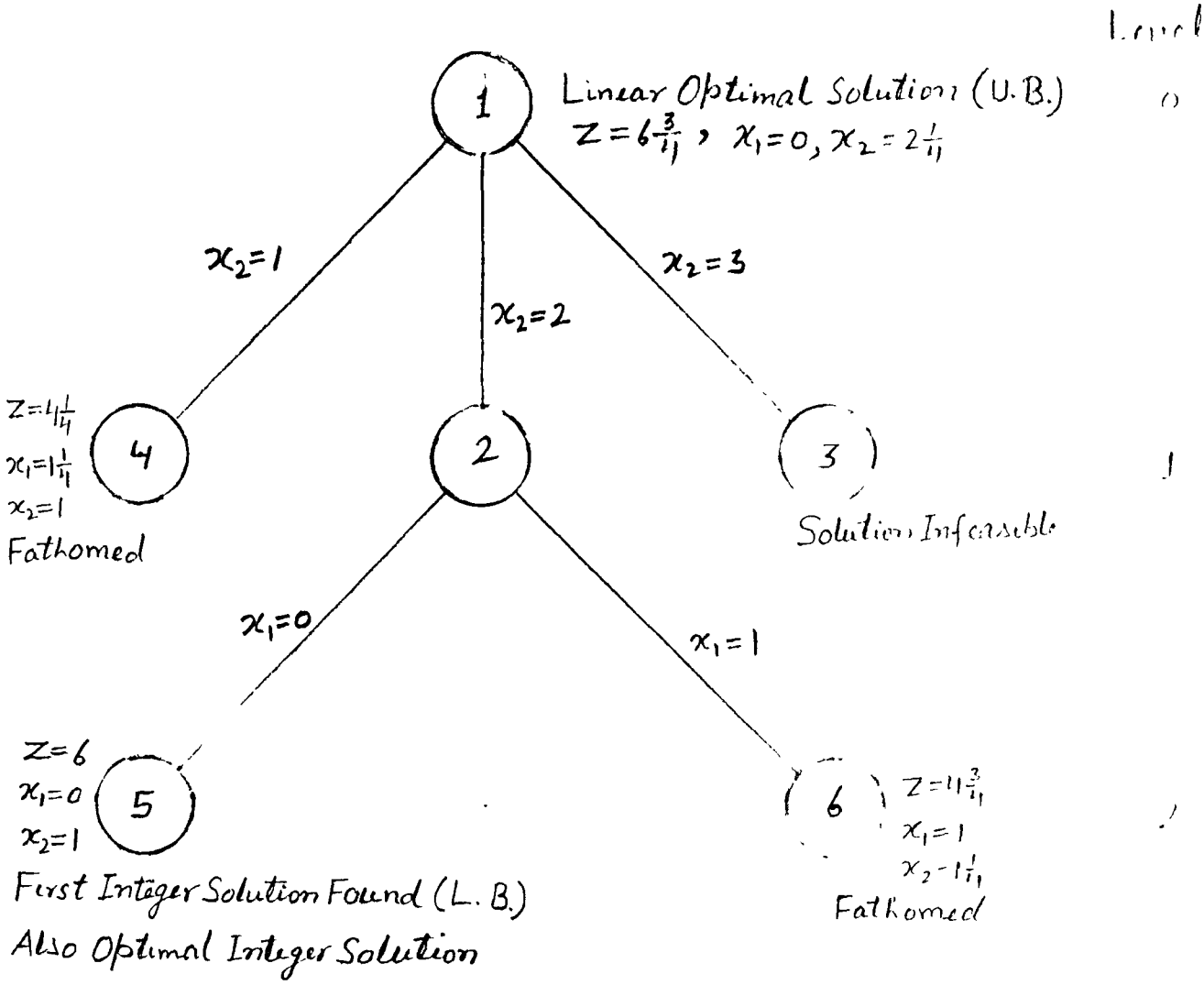


Figure 3.3 Land and Doig Algorithm

3.4.3 Dakin's Variation of the Basic Approach: (Dakin [1965])

Each time in the Land and Doig algorithm a nde is created and labeled dangling, its number, optimal linear programming solution, and other parameters have to be kept. As many dangling nodes are possible, this may mean that the technique, when computer implemented, would involve excessive storage requirements. To alleviate this difficulty, Dakin [1965] suggested that only two nodes be created from each dangling one. If an optimal linear programming solution has $x_k = t$, where t is not integral, then the first node is created by introducing the inequality $x_k \leq [t]$ and the second is defined by constraining $x_k \geq [t]+1$. This is in contrast to creating nodes by setting $x_k = [t]$ and $x_k = [t]+1$ and then defining other nodes to the left or right of these.

As in the Land and Doig algorithm, the initial node is the point x^0 with all variables free. The node is labeled dangling if its linear program is feasible and does not solvetthe MIP. At any point in the algorithm the dangling node $x^{\prime}$ with the largest optimal linear programming solution is selected to define two nodes at the next level. The subproblem at each of

these nodes is solved. The largest of these two solutions is z', an upper bound for any solution to the MIP found from x^{ℓ}. A newly created node is labeled dangling if its optimal linear programming solution exceeds the current best solution z^* for the MIP. If an improved mixed integer solution is found, z^* is updated, and those dangling nodes with an optimal subproblem solution not exceeding it are dropped from the list of dangling nodes. The procedure terminates when the list of dangling nodes is empty. Therefore, except for notation modifications, the creation of only two nodes from each selected one, the Dakin variation is essentially the same as the Land and Doig algorithm.

The Dakin algorithm converges, since in the worst case nodes could be created until the permitted range of all integer variables is reduced to zero, in which case they must take on integer values.

3.4.4 Numerical Example : The problem is

$$\begin{aligned} \text{maximise} \quad & x_1 + 3x_2 = z \\ \text{Subject to} \quad & 3x_1 + x_2 \leq 5 \\ & 4x_1 + 4x_2 \leq 9 \\ & x_1, x_2 \geq 0, \text{ integer} \end{aligned}$$

Dakin's Approach :

At the first node (x_1, x_2) the optimal linear programming solution with non-negative slacks x_3 and x_4 is given in tableau 1

Tableau-1

Node 1(x_1, x_2)

	1	$-x_1$	$-x_4$	
z	$6\frac{3}{4}$	2	3/4	
x_1	0	-1	0	
x_2	$2\frac{1}{4}$	1	1/4	$z = 6\frac{3}{4}$
x_3	$2\frac{3}{4}$	2	-1/4	$x_1 = 0$
x_4	0	0	- 1	$x_2 = 2\frac{1}{4}$

Node 2 is created by introducing the inequality $x_2 \leq [2\frac{1}{4}] = 2$ and node 3 is by constraining $x_2 \geq [2\frac{1}{4}] + 1 = 3$. As $x_2 \leq 2$ is equivalent to $\bar{x}_2 = 2 - x_2 = -\frac{1}{4} + x_1 + \frac{1}{4} \geq 0$ we commence the node 2 linear program with tableau 2(a). Similarly, since $x_2 \geq 3$ is the same as $\bar{x}_2 = -2 + x_1 = -3/4 - x_1 - 1/4 \geq 0$. This inequality shows that after adding it to tableau 1 the linear program at node 3 becomes infeasible. We then have the usual pivoting on the circled entries at node 2.

Tableau-2(a)

Node 2($\leq x_1$,2)		1	$-x_1$	$-x_4$
	z	$6\frac{3}{4}$	2	3/4
	x_1	0	-1	0
	x_2	$2\frac{1}{4}$	1	1/4
	x_3	$2\frac{3}{4}$	2	-1/4
	x_4	0	0	- 1
$2-x_2=\bar{x}_2$		-1/4	(-1)	-1/4

Tableau-2(b)

	1	$-\bar{x}_2$	$-x_4$
z	$6\frac{1}{4}$	2	1/4
x_1	1/4	-1	1/4
x_2	2	1	0
x_3	$2\frac{1}{4}$	2	-3/4
x_4	0	0	-1
$\bar{x}_2$	0	-1	0

On pivoting circled entries of tableau 2(a) we get tableau 2(b)

$z = 6\frac{1}{4}$

$x_1 = 1/4$

$x_2 = 2$

Since the solution at node 3 is infeasible, only node 2 is dangling. Node 4 is created by constraining $x_1 \leq [\frac{1}{4}] = 0$,

node 5 is defined by introducing the inequality $x_1 \geq [\frac{1}{4}]+1 = 1$. As $x_1 \leq 0$ is equivalent to $\bar{x}_1 = 0 - x_1 = -\frac{1}{4} - \bar{x}_2 + \frac{1}{4} x_4 \geq 0$ we commence the node 4 linear program with tableau 4(a). Similarly $x_1 \geq 1$ is the same as $\bar{x}_1 = -1 + x_1 = -\frac{3}{4} + \bar{x}_2 - \frac{1}{4} x_4 \geq 0$, the node 5 linear program is solved starting with tableau 5(a).

Tableau-4(a)

Node 4($\leq 0, x_2$)		1	$-\bar{x}_2$	$-x_4$
	z	$6\frac{1}{4}$	2	1/4
	x_1	1/4	-1	1/4
	x_2	2	1	0
	x_3	$2\frac{1}{4}$	2	-3/4
	x_4	0	0	-1
	$\bar{x}_2$	0	-1	0
$-x_1 = \bar{x}_1$		-1/4	1	(-1/4)

Tableau-4(b)

	1	$-\bar{x}_2$	$-\bar{x}_1$
z	6	3	1
x_1	0	0	1
x_2	2	1	0
x_3	3	-1	-3
x_4	1	-4	-4
$\bar{x}_2$	0	-1	0
$\bar{x}_1$	0	0	-1

On pivoting circled entries of tableau 4(a) we get tableau 4(b)

$z = 6$

$x_1 = 0$

$x_2 = 2$

(Integer optimal solution)

Tableau-5(a)

Node $5(\geq 1, x_2)$		1	$-\bar{x}_2$	$-x_4$
	z	$6\frac{1}{4}$	2	1/4
	x_1	1/4	-1	1/4
	x_2	2	1	0
	x_3	$2\frac{1}{4}$	2	-3/4
	x_4	0	0	- 1
	$\bar{x}_2$	0	-1	0
$-1 + x_1 = \bar{x}_1$		-3/4	(-1)	1/4

Tableau - 5(b)

	1	$-\bar{x}_1$	$-x_4$
z	$4\frac{3}{4}$	2	3/4
x_1	1	-1	0
x_2	$1\frac{1}{4}$	1	1/4
x_3	3/4	2	-1/4
x_4	0	0	-1
$\bar{x}_2$	3/4	-1	-1/4
$\bar{x}_1$	0	-1	0

On pivoting circled entry of tableau 5(a) we get tableau 5(b)

The solution at node 4 is all integer and thus first integer solution is found and fixed as lower bound. Since the value of z at node 5 is less that the lower bound, node 5 is not lebeled dangling and thus the solution at node 5 is fathomed. Node 4 gives the improved mixed integer solution i.e. $z = 6$, $x_1 = 0$ and $x_2 = 2$. As no nodes are dangling it is optimal. The tree representing the computations appears in Fig.3.4 Comparing the tree in Fig.3.3 with this one indicates that for this problem the Dakin variation is more efficient.

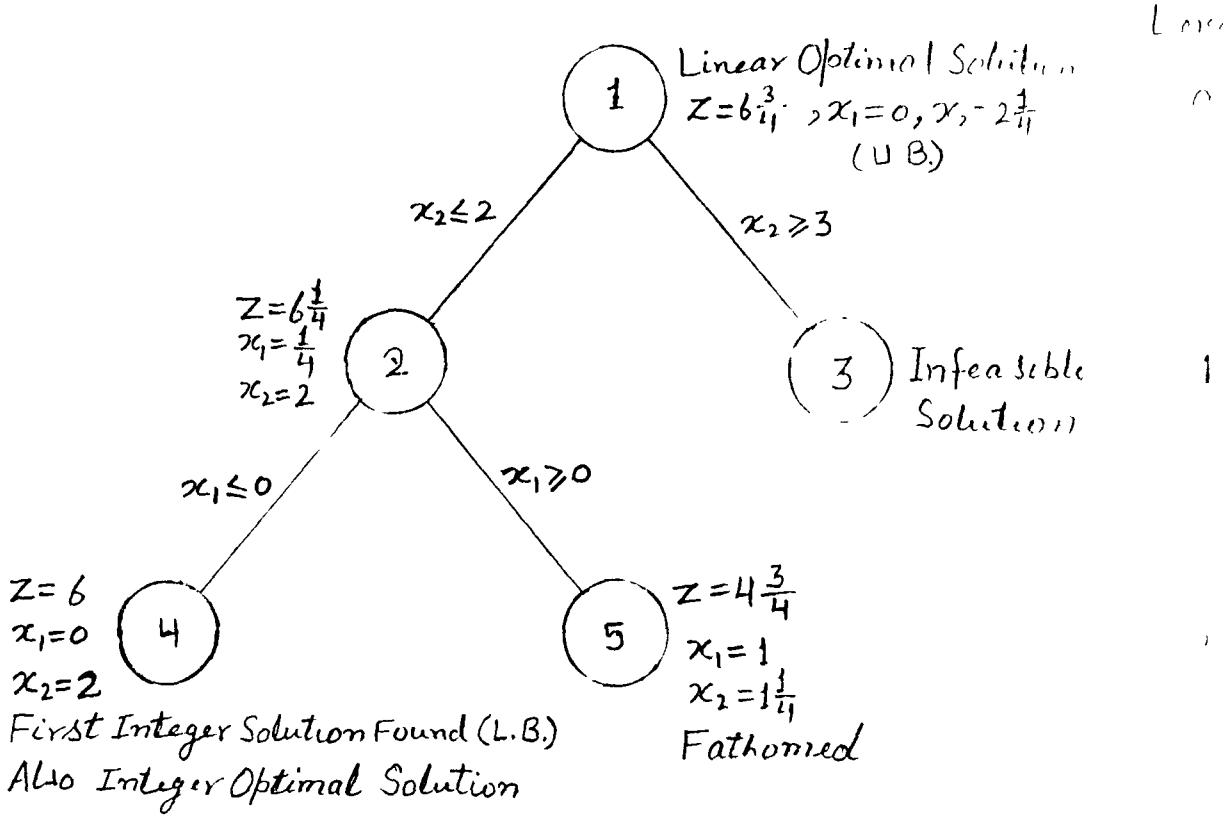


Figure 3.4 Dakin's Variation

3.5 BRANCHING RULES AND PENALTIES

At the selected dangling node, the optimal linear programming solution usually has several integer variables which are not integral. In the Land and Doig algorithm one of these variables is to be fixed at the next lowest and highest integer to define two nodes. The question is which of these variables should be picked. Up to this time we have arbitrarily picked the one with the largest fractional component. Another possibility would be to round each integer variable that is not integral to the next lowest and highest integers solve the subproblem at each of these pairs of nodes, and select the variable that produces the node with the largest optimal linear programming solution. Although this procedure probably tends to reduce the number of nodes which are explicitly examined, the additional computations will almost surely make it unworthwhile.

The optimal linear programming tableau can also be used to devise branch selection rules. A not satisfied constraint can be derived for each integer variable that is not integral. Then, by performing one dual simplex pivot, we may select a variable according to the size of the decrease in the objective

function. More precisely, suppose the optimal solution has the basic integer variable $x_k = n_k + f_k$, where n_k is a non-negative integer and f_k is a proper fraction. As either $x_k \leq n_k$ or $x_k \geq n_k + 1$, we may introduce the inequality

$$x_k^D = -x_k + n_k \geq 0 \quad \text{or} \quad x_k^U = x_k - n_k - 1 \geq 0$$

to the optimal simplex tableau. If $x_k = (n_k + f_k) + \sum_j a_{kj}(-x_{J(j)})$, where the $x_{J(j)}$'s are the current non-basic variables, then

$$x_k^D = -f_k + \sum_j (-a_{kj})(-x_{J(j)})$$

and

$$x_k^U = (f_k - 1) + \sum_j a_{kj}(-x_{J(j)}).$$

Since $-f_k$ and $f_k - 1$ are negative, the x_k^D and x_k^U equality may serve as a pivot row in the dual simplex method. After one pivot on an element in the x_k^D row (Fig.3.5), the objective function a_{oo} decreases by $-f_k(a_{oq}/a_{kq})$, where

$$\frac{a_{oq}}{a_{kq}} = \underset{j, -a_{kj}<0}{\text{minimum}} \frac{a_{oj}}{a_{kj}},$$

or, subsequently to a pivot on a coefficient in the x_k^U row, a_{oo} decreases by $(f_k - 1)(a_{or}/a_{kr})$, where

$$\frac{a_{or}}{a_{kr}} = \underset{j, a_{kj} < o}{\text{minimum}} \frac{a_{oj}}{a_{kj}} .$$

Let us define

$$P_k^D = -f_k(a_{oq}/a_{kq}), \text{ and } P_k^U = (f_k - 1)(a_{or}/a_r)$$

then P_k^D (P_k^U) can be thought of as the smallest penalty for rounding x_k down (up). Notice that P_k^D and P_k^U are nonnegative. As x_k must be integer, the sum of a_{oo} and minus the minimum of P_k^D and P_k^U is an upper bound for any mixed integer programming solution found from the selected nodes. Thus to produce an improved mixed integer solution we must have

$$a_{oo} - \underset{j, f_j > o}{\text{maximum}} (\text{minimum } (P_j^D, P_j^U)) > z^*, \qquad \ldots(3.2)$$

where z^* is the current best solution and the maximum is taken over those basic integer variables which are not integral. If (3.2) does not hold, the node should not remain dangling.

If it is possible to produce a better solution, the variable with the smallest associated penalty could be chosen. This branching rule will hopefully create a path to a " good"

mixed integer solution. However, when several $a_{oj} = o$ $(j \neq o)$, a_{oq} and a_{or} is zero and there are many ties for the smallest penalty. In this case the criterion breaks down and the choice is some what arbitrary. At the other extreme, we may select the variable which gives the largest penalty. The intent here is to eliminate the current node from consideration as quickly as possible.

3.6 PRACTICAL CONSIDERATIONS IN USING BRANCH AND BOUND

At first it might seem that the branch and bound algorithm is rather cumbersome, but the underlying logic is actually quite simple. The preceeding exposition examines the workings of the algorithm in minute detail, to clarify the reasoning and convince the reader of its correctness. Once the logic of the algorithm is understood, this degree of introspection is no longer necessary, and the solution process can be performed much more quickly. After a little practice the method can be used to solve small problems quite easily by hand, provided the subproblem solutions are easy to obtain. It is also straightforward to program a computer to implement the algorithm, although some care is required in choosing a sensible data structure to

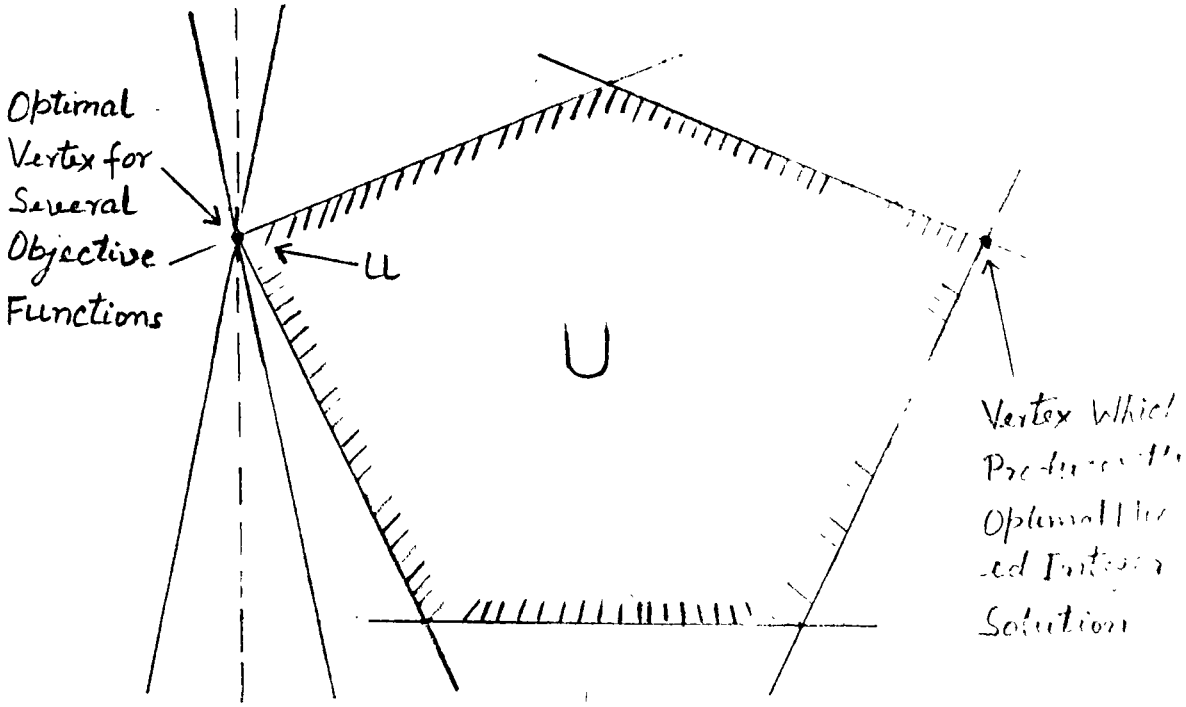

Optimal
Vertex for
Several
Objective
Functions
U
U
Vertex Which
Produces the
Optimal Mixed Integer
Solution

Figure 3.5

represent the branching diagram.

In many applications it might be satisfactory to stop the branch-and-bound algorithm when a solution has been found that is not optimal but only sufficiently close. Thus, if at some stage in the solution process the upper bound is slightly higher than the lowest objective value (lower bound) among the nodes that remain unfathomed, little further improvement can be obtained, and the current incumbent solution might be declared closed enough to optimal.

In the example problem we were able to fathom many of the nodes as the solution progressed, so that total number of nodes we needed to consider was quite small. Unfortunately, this does not always happen. Depending on the data of a problem, it can turn out that many branch-and-bound iterations are performed without fathoming many nodes, leading to a dramatic growth in the number of subproblems to be solved. In fact, the total amount of work needed to solve an integer program can grow exponentially with the size of the problem (which depends on the number of variables and the number of values they can take on). This phenomenon is thought to be possible no matter what algorithm

is used, and makes integer programming fundamentally more difficult than linear programming. Although the behaviour of problems from real applications is never as bad as it could be in the worst case, the proliferation of subproblems imposes a practical limit on the size of problems that can be solved on a computer of given memory capacity and speed.

Usually, most of the computational work in the branch-and-bound algorithm is devoted to solving the subproblems. In the preceeding example we ignored this point simply providing the solutions of the subproblems at each stage, but the work in bounding step of the algorithm is typically much greater than the work involved in deciding how to branch, in fathoming nodes, and in checking for unfathomed nodes. This is particularly true when the subproblems are nonlinear programming programs.

3.7 PARTITIONING IN MIXED INTEGER PROGRAMMING

Using linear programming in duality theory, it is possible to show that any mixed integer program can be written as an integer program. This suggests solving a mixed integer problem by solving is equivalent integer program. However, it is often computationally impossible to obtain all the constraints of the

integer program since each requires the numerical value of an extreme point or an extreme ray in a convex polyhedron, and there are as many constraints as there are extreme points and extreme rays. As the polyhedron may have a vast number of extreme points, all the constraints can not usually be listed. To overcome this difficulty, J. F. Benders [1962] proposed a technique in which the equivalent integer program is solved after generating only a subset of its constraints. That is, the remaining " implicitly enumerated " constraints do not further constrain the integer program. The " partitioning algorithm " works by successively solving a linear program and an integer program. The linear program produces an extreme point or an extreme ray and a single constraint for the integer program. Also, the value of its optimal solution gives an upper bound for the optimal solution to the mixed integer program. When solved, the integer program, which is mixed integer program's equivalent when it has all its constraints, yields a nondecreasing lower bound. When the two bounds coincide, the optimal mixed integer solution has been found and the process terminates.

Before developing the procedure, we note that Benders' [1962] algorithm is applicable to a more general problem. In particular,

it solves any mathematical program in which the variables can be partitioned into two subsets, so that when the variables in one are assigned numerical values the problem reduces to a linear program. A mixed integer program, of course, fits this description and is the only type of problem that we will consider. Nevertheless, an outline of the general development appears in Lasdon [1970]. We also note that one of the first to specialize Bender's algorithm for the mixed integer program is Balinski [1965], and an application of the specialized version to the uncapacitated plant location problem may be found in Balinski and Wolfe [1963]. A special purpose enumerative technique for the integer programs that appear during the partitioning algorithm is explained in Balas [1970]. Related work is in Rardin and Unger [1973] and in Rutten [1973].

3.7.1 Bender's Partitioning Algorithm : (Bender [1962])

As we have already mentioned, the polyhedron U usually has a vast number of extreme points, and therefore it is virtually impossible to generate all of the constraints of the integer program as given below :

$$
\begin{array}{lll}
\text{I} \quad \text{minimize} & s & \\
\text{subject to} & s \geq cx + u^p (b - Ax) & (p=1,\ldots,P) \\
& o \geq v^s (b - Ax) & (s=1,\ldots,S) \\
\text{and} & x \geq o \text{ and integer} &
\end{array}
$$

However, its derivation suggests a procedure for solving the mixed integer program. Note that for a fixed non-negative integer vector x we can solve the linear program DL. As this problem is more restricted then the mixed integer program, the value of its maximal solution when added to cx is an upper bound on the optimal solution to the mixed integer program. (If the problem DL has an unbounded solution, its value can be thought of as infinite). Furthermore, problem DL, when solved, reveals an extreme point of U or the direction of one of its extreme rays, and thus an inequality for the integer program I. Suppose the integer program is solved with this one constraint. Its optimal solution is a lower bound on the best solution to the mixed integer program. Also, it yields a non-negative integer vector x for problem DL, which, when solved, produces a new extreme point or ray direction and thus a second inequality for the integer problem. The integer program, now with two constraint

is solved and an at-least-as-good lower bound and non-negative integer vector x is found. With the new value for x, the linear program DL can again be solved, and so forth. The process terminates when the lower bound equals the upper bound. To find the optimal y vector, problem L is solved with x equal to its optimal value. The procedure is outlined below:

Step 1- (Initialization) Select a non-negative integer value for the vector x, say $\bar{x}$, and set $z^u(z^\ell)$ arbitrarily large (small). Go to step 2.

Step 2- (Linear Programming Phase) Solve the linear program

$$\begin{aligned} \text{DL} \quad & \text{maximize} \quad u(b - Ax) \\ & \text{subject to} \quad u \in U = \{u \geq o \mid uD \leq d\}, \end{aligned}$$

with x fixed at $\bar{x}$. This yields an optimal extreme point u^p or extreme ray direction v^s. In the extreme point situation. Replace the value of z^u by $c\bar{x} + u^p (b - Ax)$ whenever $z^u > c\bar{x} + u^p (b - Ax)$. Go to step 3.

Step 3- (Integer Programming Phase) Solve the integer program

$$\begin{aligned} \text{I} \quad & \text{minimize} \quad z \\ & \text{Subject to} \quad z \geq cx + u^p(b-Ax) \qquad (p=1,\ldots,p^\ell) \\ & \qquad\qquad o \geq v^s (b - Ax) \qquad (s=1,\ldots,s^\ell) \\ & \text{and} \qquad\quad x \geq o \text{ and integer} \end{aligned}$$

where $P^{\ell}(S^{\ell})$ is the number of extreme points (extreme rays) found from Step 2. Set z^{ℓ} equal to the minimal value of z and $\bar{x}$ equal to the optimal x vector. Go to step 4.

Step 4- (Termination test) If $z^{\ell} < z^{u}$, go to step 2. Otherwise ($z = z^{u}$), $\bar{x}$ is optimal. In this case, solve problem L

$$\begin{aligned} &\text{minimize} && dy \\ &\text{Subject to} && Dy \geq b - Ax \\ &\text{and} && y \geq 0 \end{aligned}$$

to obtain the optimal y vector $\bar{y}$. Then $x = \bar{x}$, $y = \bar{y}$, and $z^{\ell} = z^{u} = c\bar{x} + d\bar{y}$ solves the mixed integer program-terminate.

3.7.2 Properties of the Partitioning Algorithm :

1. In the next section, we show that unless the optimal mixed integer solution is found,

(i) an optimal solution to the integer program in Step 3 never repeats itself.

(ii) each time the linear program DL in Step 2 is solved, a new extreme point (or extreme ray) of U is generated.

2. Unless the bounding inequality $z \geq z^{\ell}$ is explicitly introduced to each integer program, all generated non-redundant

" s inequalities " have to be kept. In particular, an inequality $s \geq cx + u^p (b - Ax)$ can not be dropped from the integer program in Step 3 if its slack variable is positive in an optimal integer programming solution. This follows, since an inequality may have eliminated integer points which correspond to lower value of s. Thus, if such an inequality is dropped, optimal solutions to subsequent integer programs could yield worse lower bounds and thus the algorithm may not converge.

3. Every time an integer vector x yields an optimal vector u for problem DL, a mixed integer solution, and thus an upper bound for the minimal mixed integer solution, is found. Therefore, if computer time or storage becomes excessive, computations can be stopped and the current best solution s^u can be used. Notice that it is at most $s^u - s^\ell$ away from the true optimum.

4. The partitioning aspect of the algorithm preserves the structure of the matrix D in the linear program L, and hence in its dual problem DL. Thus, for example, if D is a transportation type constraint matrix, then every occurrence of problem DL involves the dual of a transportation problem.

5. The computational efficiency of the algorithm depends principally on the number of extreme points and extreme rays that have to be found. If this number is relatively large, the approach will not be efficient.

6. The partitioning algorithm has a cutting plane flavor. In particular, each time an increased lower bound is obtained from the integer program, the most recently added z inequality cuts off the previous optimal integer solution.

3.7.3 Application of the Partitioning Algorithm to the Uncapacitated Plant Location Problem : (Balinski [1963])

Balinski and Wolfe (1963) presented a paper " On Benders Decomposition and a Plant Location Problem". The special structure of the uncapacitated plant location problem permits us to obtain, very simply, the optimal solutions to the linear programs L and DL that appear during Bender's partitioning algorithm. The uncapacitated plant location problem (A slightly different version of the uncapacitated plant location problem is discussed in Balinski and Wolfe [1963]) is

$$\text{minimize} \quad \sum_{i=1}^{m} f_i x_i + \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} y_{ij}$$

$$\text{subject to} \quad n_i x_i - \sum_{j=1}^{n} y_{ij} \geq 0 \qquad (i=1,\ldots,m)$$

$$\sum_{i=1}^{m} y_{ij} = 1 \qquad (j = 1,\ldots,n)$$

$$y_{ij} \geq 0 \qquad \text{for all } i,j$$

$$\text{and} \quad x_i = 0 \text{ or } 1 \qquad (i=1,\ldots,m)$$

where f_i is a mixed cost associated with plant i, x_i is 1(0) if plant i is open (closed), c_{ij} is a non-negative shipping cost, y_{ij} is the fraction of customer's j demand satisfied by plant i, n_i is the number of customers that can be satisfied by plant i (which for ease of exposition will be taken as n), and there are m plants and n customers.

Notice that for a fixed zero-one vector x, problem P1 reduces to the linear problem L. Its dual is problem DL.

L $\quad$ minimize $\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} y_{ij}$

$$\text{Subject to} \quad -\sum_{j=1}^{n} y_{ij} \geq -n_i x_i \qquad (i=1,\ldots,m)$$

$$\sum_{i=1}^{m} y_{ij} \geq 1 \qquad (j=1,\ldots,n)$$

$$\text{and} \quad y_{ij} \geq 0 \qquad \text{for all } i,j$$

DL $\quad$ maximize $\sum_{j=1}^{n} v_j - \sum_{i=1}^{m} n_i x_i u_i$

$$\text{Subject to} \quad v_j - u_i \leq c_{ij} \qquad (i=1,\ldots,m \text{ and } j=1,\ldots,n)$$

$$u_i \geq 0 \qquad (i=1,\ldots,m)$$

$$\text{and} \quad v_j \text{ unrestricted} \qquad (j=1,\ldots,n)$$

Now consider the convex polyhedron

$$U = \{(u_1,\ldots,u_m,v_1,\ldots,v_n)/v_j-u_i \leq c_{ij} \quad i,j,$$

$$\text{and} \quad u_i \geq 0 \quad \text{for } i = 1,\ldots, m\}$$

It is independent of x and has P extreme points (S extreme ray direction), each having components $(u_i^P, v_j^P)((u_i^S, v_j^S))$. By analysing we have that the mixed integer problem P1 is equivalent to the integer program

I minimize z

$$\text{Subject to} \quad z \geq \sum_{i=1}^{m} f_i x_i + \sum_{j=1}^{n} v_j^P - \sum_{i=1}^{m} n_i x_i u_i^P, \quad (p=1,\ldots,P)$$

$$0 \geq \sum_{j=1}^{n} v_j^S - \sum_{i=1}^{m} n_i x_i u_i^S \qquad (s=1,\ldots,S)$$

$$\text{and} \quad x_i = 0 \text{ or } 1 \qquad (i=1,\ldots,m)$$

Remember that the second set or extreme ray inequalities are necessary and sufficient conditions on x to admit feasible solutions $y=(y_{ij})$ to problem L and thus problem P1. However, a careful examination of the linear program L shows that for any zero-one vector x, except $x=0$, there is a feasible solution y. That is if atleast one plant is open, all the customers can be serviced. Thus, the ray inequalities can be dropped so long as $x \neq 0$ is enforced.

CHAPTER-IV

SEARCH ENUMERATION

4.1 INTRODUCTION

As in branch and bound, search algorithms either explicitly or implicitly enumerate all possible integer vectors. The literature devoted to search techniques is confined principally to the zero-one integer program and, in some instance, to the zero-one mixed integer program. Many integer programming problems that arise in practical settings have the special property that some or all of their variables are restricted to take on only the values 0 or 1. Such variables are called 0-1 variables or binary variables, and they often arise naturally in the formulation of problems that involve yes or no decisions. If an integer programming problem has only 0-1 variables, the bounding step in the branch and bound algorithm can be simplified considerably. Because most of the work of the branch and bound algorithm is in the bounding step, this simplification can make the algorithm very much faster.

An enumerative procedure unlike branch and bound algorithm introduced by Land and Doig [1960], was first proposed by

Egon Balas [1963]. The technique, referred to as the "additive algorithm", is described more fully by Balas [1965]. This work was elaborated on by Glover [1965], and later by Lemke and Spielberg [1967]. Linear programming, absent in earlier search algorithms, was computer implemented by Geoffrion [1969] and by Slakin and Spielberg [1968]. Papers which describe implicit enumeration tests include, the linear programming post optimization work of Driebeek [1966], which was implemented by Shareshian [1966], the implementation and uses of "surrogate constraints" by Geoffrion [1969] and Balas [1967], which was introduced by Glover [1965] and surveyed by him in [1968], the generation and use of other inequalities implied by the constraint set by Bradley, Hammer, and Wolsey [1973], Johnson and Spielberg [1971], and by Spielberg [1972], [1972], and a refinement of the original Balas tests by Glover and Zionts [1965] and Lemke and Spielberg [1967]. Relaxing the originally rigid branching rules, suggested by Glover and Zionts [1965], was formally proposed by Spielberg [1968] and later by Guingnard and Spielberg [1968], [1971], [1971], and Tuan [1971]. Allowing the enumeration to start at a point different from zero appeared

in Salkin [1970], [1969], Salkin and Spielberg [1968], and Spielberg [1969]. A general discussion of the additive algorithm is in Geoffrion [1967], and somewhat similar scheme are discussed in Lawler and Bell [1966] and in Golomb and Baumert [1965]. Computational results for general zero-one problems appear in Fleischman [1967], Freeman [1966], Geoffrion [1969], Jambekar and Spielberg [1973], Lemke and Spielberg [1967], Petersen [1967], and Salkin [1970]. Survey articles which discuss the additive algorithm and extensions include Balinski and Spielberg [1969], Geoffrion and Marsten [1972], and Salkin [1973].

Transformation of nonlinear programming problems to zero-one linear programming problems is found in Balas [1984]. Cartisensen [1985] discusses intermediate feasibility in 0-1 integer linear systems. Duality in Polynomial 0-1 Optimisation is given in Lu and William [1987].

We begin by describing the problem and reviewing the method of Balas [1965].

The zero-one integer linear programming problem may be formulated as follows :

$$\text{Minimize} \quad cx = s$$

$$\text{Subject to} \quad Ax \leq b \,, \; o \leq x \leq e \qquad \ldots(4.1)$$

$$\text{and} \quad x_j \text{ integer} \quad (j=1,\ldots,n)$$

where A is an m by n matrix and e is a column of ones.

4.2 THE ADDITIVE ALGORITHM OF BALAS

Balas [1965] sets up all the constraints as inequalities of the form

$$Ax \leq b \quad \text{or} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

He achieves this form by multiplying inequalities with the reverse inequality by -1, and using two inequalities to represent an equality. Then, if all $b_i \geq o$, the optimal solution is $x_j = o$, $j = 1,\ldots,n$. We shall present the algorithm assuming that a partial solution $(j_1,\ldots, j_p)$ is available, noting that the initial partial solution has no variables assigned. Then the following procedure is used :

Step I(a)- If the current partial solution satisfies the problem constraints, complete the partial solution by setting all free variables to zero. Set z^* , the current minimal integer solution = the objective function value of the solution

and save the solution. No other completion of the partial solution can be better than the present one, hence, no other continuations need be examing. Go to step V .

(b)- If the partial solution does not satisfy the problem constraints, go to step II. Denote the objective function value of the current partial solution as s^l.

Step II- For free variables x such that $s^l + c_j \leq s^*$ (s^* is initialized at ∞ if no feasible solution is known initially), and such that some coefficient $a_{ij} < 0$ for atleast one constraint

$$b_i - \sum_{k=1}^{p} a_{ijk} x_{jk} < 0$$

(we shall denote the set of such variables as the set N), check the relationships

$$\sum_{j \in N} \bar{a_{ij}} \leq b_i - \sum_{k=1}^{p} x_{jk} e_{jk} (< 0) \qquad \ldots(4.2)$$

An intuitive interpretation of (4.2) is that by restricting attention to variables which can help satisfy [illegible] constraints, we may check to see whether any constraints can never be satisfied. If any such relationship (4.2) is violated,

there is no feasible continuation, hence, go to step V.

Otherwise, go to step III

Step III-(a) If all relationships (4.2) are satisfied as strict inequalities, determine which free variable (if set to 1) of the set N would reduce the total infeasibility (the sum of the absolute values of the amount by which all the constraints are violated) most, i.e. choose $j_{p+1} \in N$ and set $x_{j_{p+1}} = 1$ such that

$$\sum_{i=1}^{m} \left(b_i - \sum_{k=1}^{p+1} a_{ijk}\, x_{jk} \right)$$

is maximised. The new partial solution is $(j_1, \ldots, j_p, j_{p+1}^{++})$. Go to step II.

(b)- If, for any subset of constraints, the relationship (4.2) holds as equalities, denote by the set F all free variables x_j such that $a_{ij} < o$ for at least one constraint of the subset. Check the relationship

$$\sum_{j \in F} c_j < z^* - z' \qquad \ldots(4.3)$$

Then go to step IV.

Step IV-(a) If (4.3) is satisfied, then $x_j = 1$, $j \in P$ is the only possible optimal feasible continuation of the present partial solution. It may or may not be feasible, however. The next partial solution to test is therefore $(j_1,\ldots,j_p,j_{p+1}^+,\ldots, j_{p+q}^+)$, where $j_{p+1},\ldots, j_{p+q} \in P$. Go to step 1.

(b)- If (4.3) is not satisfied, then there is no possible optimal feasible continuation of the present partial solution. Go to step V.

Step V-(a) Consider the present partial solution. Find the rightmost element j_k^{++} and delete all elements to the right of it. Replace it by j_k^-. That is the new partial solution. Go to step 1.

(b)- If there is no element j_k^{++} in the present partial solution, the (implicit) enumeration is complete and the optimal solution (which has been saved in step 1a) has objective function value z^*. If z^* is ∞, there is no feasible solution.

Remarks- The above algorithm is essentially the one presented by Balas [1965]. Note that it yields only one optimum (the algorithm could readily be altered to yield all optima, as Balas points out). Note also that we have made no attempt to implement alterations to the algorithm, such as those indicated

by Glover and Zionts [1965], among others. In addition, a notation different from that of Balas has been used here in an attempt to simplify the presentation.

4.3 A GENERALIZED ADDITIVE ALGORITHM (Zionts [1972])

Generating upper and lower bounds on the variables, together with Balas structure of implicit enumeration, and by taking advantage of simplification that can be obtained, a simpler and more powerful algorithm may be achieved. The resulting algorithm includes special tests developed by other authors, including Fleishmann [1967] and Geoffrion [1966]. Conceptually, it is convenient to pose the problem in Balas framework with a few alterations :

1. Equality constraints are represented as such
2. A constraint of the form

$$\sum_{j=1}^{n} c_j x_j \leq z^* - \epsilon \quad \text{where} \quad \epsilon \leq C \leq \min_j \{c_j\}$$

is added where z^* is minimum objective function for a feasible solution found thus far. The effect of this constraint is to imbed all checks involving the objective function within the general constraint checking procedure. If $C = \epsilon$ is used, then

all alternate optima will be found. Initially, $z^* = M$ (or ∞) (where M is sufficiently large number, e.g. $\sum c_j$).

3. Replace steps I, II and IV of Balas' method by the following:

2'. Use the results of theorem given at the end of this algorithm to generate upper and lower bounds, as ap ropriate, for each zero-one variable in every constraint.

(a) If a lower bound greater than zero (but less than or equal to one is found for some variable x_k, then x_k is implied to be one in all continuations of the present partial solution. Thus k+ augments the current partial solution. Go to step .

(b) If a lower bound greater than one is found for some variable x_k, then there is no feasible continuation. Go to step V.

(c) If an upper bound less than one but not less than zero is found for some variable x_k, then x_k is implied to be zero in all continuations of the present partial solution. Then K-augments the current partial solution. Go to step .

(d) If an upper bound less than zero is found for some variable x_k, then there is no feasible continuation. Go to step .

(e) If all upper bounds are atleast one and all lower bounds

are at most zero, then no tighter bounds are available. Go to step 3'.

3'- Determine which free variable would reduce the total infeasibility (the sum of the absolute values of the amount by which all constraints are violated) most, i.e. choose j_{p+1} and set $x_{j_{p+1}} = 1$ such that

$$\sum_{i \notin E} \left| \left(b_i - \sum_{k=1}^{p+1} a_{ijk}\, x_{jk} \right)^- \right| + \sum_{i \in E} \left(\sum_{k=1}^{p+1} a_{ijk}\, x_{jk} - b_i \right) \Bigg|$$

is minimized where E is the set of equalities. The new partial solution is $(j_1, \ldots, j_p, j_{p+1}{}^{++})$. Go to step 1. (This is equivalent to Balas' choice step, except that equalities are represented as such.)

4'- Step 4 of Balas' method is deleted.

One refinement worth mentioning is the use of alternate criteria in step 3'. Two promising possibilities are :

1. Choosing the free variable with the minimum c_j.

2.(a) Where the total infeasibility can be reduced (see step 3'), choose the variable with the minimum cost per unit of infeasibility reduction.

(b) Where the total infeasibility can not be reduced, use the rule given in 1 above.

The first possibility is computationally inexpensive, whereas the second is computationally expensive. The second might be used in the early stages of computation until an integer solution is found. Then one of the other strategies may be used.

Actually, the calculations in step 2' can be simplified tremendously by elimination of tests which cannot occur, thus avoiding repititions calculations. (Professor Earl McCoy, University of Alabama, has suggested some of the ideas for the simplification) (only two variables in each constraint are checked.)

4.3.1 Theorem : Upper and Lower Bounds for Integer Variables x_j are given as below :

1. For $a_{ij}^{+} > 0$, a lower bound for integral x_j is given by

$$h_j = \min \left\{ 0, \frac{b_i}{a_{ij}^{+}} - \frac{1}{a_{ij}^{+}} \sum_{k \neq j} a_{ik}^{+} h_k - \frac{1}{a_{ij}^{+}} \sum_{k \neq j} a_{ik}^{-} h_k \right\}$$

And an upper bound for integral x_j is given by

$$u_j = \left[\frac{b_i}{a_{ij}^+} - \frac{1}{a_{ij}^+} \sum_{k \neq j} a_{ik}^+ h_k - \frac{1}{a_{ij}^+} \sum_{k \neq j} a_{ik}^- u_k \right]$$

2. For $a_{ij}^- < 0$, a lower bound for integral x_j is given by

$$h_j = \min \left\{ 0, \left(\frac{b_i}{a_{ij}^-} - \frac{1}{a_{ij}^-} \sum_{k \neq j} a_{ik}^+ h_k - \frac{1}{a_{ij}^-} \sum_{k \neq j} a_{ik}^- u_k \right) \right\}$$

And an upper bound for integral x_j is given by

$$u_j = \left[\frac{b_i}{a_{ij}^-} - \frac{1}{a_{ij}^-} \sum_{k \neq j} a_{ik}^+ u_k - \frac{1}{a_{ij}^-} \sum_{k \neq j} a_{ik} h_k \right]$$

We round lower bounds up and upper bounds down, if they are not integer.

4.3.2 Numerical Example : Problem 1 of Balas [1965]

$$\text{Minimise} \quad z = 5x_1 + 7x_2 + 10x_3 + 3x_2 + x_5$$

$$\begin{aligned} \text{Subject to} \quad -x_1 + 3x_2 - 5x_3 + -x_4 + 4x_5 &\leq -2 \\ 2x_1 - 6x_2 + 3x_3 + 2x_4 - 2x_5 &\leq 0 \\ x_2 - 2x_3 + x_4 + x_5 &\leq -1 \\ x_1, \ldots, x_5 &= 0,1 \end{aligned}$$

Using Balas' (1965) method, we have the following partial solutions :

1. (o). In step IIIa add 3++.
2. (3++). In step IIIa add 2++.
3. (3++,2++) s* = 17, feasible. In step V . backtrack.
4. (3++,2⁻). In step IV, relation (4.2) is violated by constraint 2, therefore, backtrack (step V).
5. (3⁻). In step IV, relation (4.2) is violated by constraint 3, therefore, backtrack (step V). $x_3 = 1$, $x_2 = 1$ is the optimal solution.

Using the generalized method, we have the following partial solutions :

1. (o). From constraint 3, the lower bound on x_3 is seen to be 1.
2. (3+). From constraint 2, the lower bound x_2 is seen to be 1.
3. (3+,2+) s = 17, feasible. No backtracking is possible. The problem is solved.

4.4 GENERALIZING THE ZERO-ONE ADDITIVE APPROACH TO ALL-INTEGER PROBLEMS

The generalization of the zero-one additive algorithm to solve all integer problems is straightforward and has been proposed by Krolak [1966] and others. (Such approaches are,

of course, all-integer, rather than zero-one integer.) All of the tests on bounds carry over, but are not additive in this case. What is required is a scheme for representing partial solutions, analogous to that used in our exposition of Balas' additive algorithm, so that only one active solution need be remembered at a time. Such a scheme might include something like the following :

1. When a variable is to be set to some value, it is set to the largest (or alternatively smallest) feasible value.
2. When backtracking and considering another value for the next previous choice variable, decrement (increment) that variable for the next partial solution. If the variable happens to be equal to its minimum (maximum) feasible value, treat it as a variable whose value is implied and backtrack to the next previous choice variable.
3. Recompute the bounds after each choice step.

Computational results using all-integer implicit enumeration algorithms have been meager to date, and the results do not appear very promising.

4.5 SURROGATE CONSTRAINTS

To accelerate the solution for zero-one problems, Glover [1965] proposed the use of surrogate constraints, a positive linear combination of constraints which yields a constraint that is " strong " in some to be defined sense. Use of surrogate constraints in a linear programming context to identify redundant constraints had earlier been considered by Thompson, Monge, and Zionts [1966]. Surrogate constraints in zero-one integer programming have been further studied and used to advantage by Balas [1967] and Geoffrion [1969]. Their methods are closely related, and we shall present a variation that seems to combine good features of each. First, we define the term " strength of a surrogate constraint " as used by Balas [1967].

4.5.1 Definition : A surrogate constaint is a constraint $y'Ax \leq y'b$, where $Ax \leq b$, is the constraint set and $y \geq o$ is a vector of appropriate order.

4.5.2. Definition: Given two surrogate constraints, $a_0x \leq b_0$, and $a_1x \leq b_1$, the stronger surrogate constraint has the larger objective function value when minimising an objective function,

subject only to the surrogate constraint and non-negativity constraints.

The following theorem (4.5.3) is a variation on those presented by Balas [1967], Geoffrion [1969], and Glover [1965].

4.5.3 Theorem : For a given linear programming problem the optimal dual solution yields multipliers for constructing a strongest surrogate constraint.

Proof : The problem is

$$\begin{aligned} &\text{Minimize} && z = c'x \\ &\text{Subject to} && Ax \leq b \end{aligned} \qquad \ldots(4.4)$$

where x is a vector of zeros and ones and the continuous analog in maximizing form is

$$\begin{aligned} &\text{Maximize} && z = -c'x \\ &\text{Subject to} && Ax \leq b \\ & && x \leq 1 \\ & && x \geq 0 \end{aligned} \qquad \ldots(4.5)$$

The dual problem is

$$\begin{aligned} &\text{Minimize} && z = b'y = 1'u \\ &\text{Subject to} && A'y + u \geq -c \end{aligned} \qquad \ldots(4.6)$$

Let an optimal solution to the dual be y^*, u^*, and its corresponding objective function value be

$$z^* = b'y^* + l'u^*$$

The corresponding surrogate constraint is

$$y^{*'}Ax + u^{*'}x \leq y^{*'}b + u^{*'}l = z^*$$

Now we must prove that the optimal solution to problem (4.7) below is minimum for $y = y^*$ and $u = u^*$.

$$\begin{aligned} &\text{Maximise} && -c'x \\ &\text{Subject to} && (y'A + u')x \leq (y'b + u'l) \qquad \ldots(4.7) \\ & && x \geq 0 \end{aligned}$$

Let x^* be the optimal solution to (4.5), then by duality we have that $-cx^* = (y^{*'}A + u^{*'})x^* = y^{*'}b + u^{*'}l = z^*$. We may also write

$$-c'x \leq (y^{*'}A + u^{*'})x \leq y^{*'}b + u^{*'}l \qquad \ldots(4.8)$$

The optimal solution to (4.7) with y^* and u^* is x^*. For any other surrogate constraint generated by any non-negative multipliers y and u, clearly any feasible solution to (4.5)

including x^* is feasible. In addition some solutions which are not feasible solutions to (4.7) in such a case can possibly be greater than $-c'x^*$, but never less, thereby proving the theorem.

We indicate briefly the approaches of Balas [1967], Geoffrion [1969], and Glover [1965], all of which are very much like the implicit enumeration scheme with surrogate constraints given above.

4.5.4 The Balas Filter Method [1967] : The Balas Filter method first constructs the surrogate constraint defined in Definition 4.1. Then, within the surrogate constraint,

$$\sum_{j=1}^{n} a_j x_j \leq b \qquad (b < 0)$$

the variables are arranged in such a manner that all a_j negative have lower indices than any a_j positive. The following conditions must hold :

1. For $a_k < 0$, $a_i < 0$, and $i > k$ we must have $c_i/a_i \leq c_k/a_k$. This implies that of any two variables that help to satisfy the surrogate constraint, the more ''efficient'' variable is first.
2. For $a_k > 0$, $a_i > 0$, $i > k$ we must have $a_k < a_i$. This

implies that of any two variables that do not help to satisfy the surrogate constraint, the one that increases infeasibility least is first.

Then, using only the rearranged surrogate constraint, the method generates a tree of partial solutions in a specified order. The best partial solution (i.e., the one with the minimum objective function value) is selected from the tree. There are two possibilities :

1. If it is not feasible with respect to the surrogate constraint, it is used to generate continuations in the tree by branching on a free variable. (If there is no feasible continuation with respect to the surrogate constraint, the branch is terminated.) Then the best partial solution remaining in the three is examined, and the process repeated.

2. If the solution is feasible with respect to the surrogate constraint, it is checked for feasibility with respect to all the constraints. If it satisfies those constraints, it is the optimal integer solution, otherwise, that solution (after first checking the complete set of constraints for variables implied to be zero or one, and for the possibility of no feasible continuation) is

extended by branching on a free variable and the new solutions are stored in the tree. Then the best partial solution remaining in the tree is examined, and the process repeated.

Balas' surrogate constraint problem for the problem of Example 4.2 is arranged in the following manner :

$$\text{Minimise} \quad z = 7x_2 + 10x_3 + x_5 + 5x_1 + 3x_4$$

$$\text{Subject to} \quad -7x_2 - 10x_3 + 3\tfrac{2}{5}x_5 + 5\tfrac{1}{5}x_1 + 14\tfrac{1}{5}x_4 \leq -9$$

(The branching proceeds from left to right). The method uses tests which are an improvement over those of the additive algorithm though not as comprehensive as those of the generalised algorithm.

4.5.5 Geoffrion's Improved Implicit Enumeration Approach [1969] :

As mentioned earlier, the exposition is most closely related to that of Geoffrion. There are two relatively minor differences in Geoffrion's approach which have not been indicated. First, Geoffrion's surrogate constraint is defined as

$$y' Ax + c'x \leq y'b + \bar{z}$$

where $\bar{z}$ is an upper bound to the objective function value of the integer optimum (y is the optimal solution to the dual, yielding the strongest constraint by his definition [slightly

different from improved one]). Second, he uses a relatively simple set of infeasibility tests in implementing his scheme, noting that any improved rules could, of course, be used.

4.5.6 Glover's Multiphase Dual Method [1965] :

Glover's Multiphase dual algorithm uses only one surrogate constraint, which is derived in a heuristic manner by choosing the constraint multipliers arbitrarily. The constraint is modified if it is found that any of the constraints which comprise it cannot possibly be binding in a continuation of the current partial solution. The multiplier of such a constraint in forming the surrogate constraint is set to zero until appropriate backtracking occurs, at which time the appropriate multiplier in the surrogate constraint is restored to its earlier value. In addition, the continuous optimal continuation of a partial solution is solved for as the problem solution proceeds—subject only to the surrogate constraint and the zero-one bound constraints. The other tests of the multiphase dual method may be seen as special cases of the generalised tests given here.

4.6 OTHER COMPUTATIONAL DEVICES

Numerous other devices have been used in zero-one integer

programming schemes. These include choosing a ''better'' origin than the dual feasible solution used by most schemes. Of course, tests such as those used by Balas and others on the objective function would not be valid in general. The refined use of bounds into which the additive algorithm was generalised earlier would still be valid. Other devices, such as indexing the variables for each constraint in the order of their desirability (i.e. the minimum value of c_j/a_{ij} $|a_{ij} \neq 0|$ is most desirable), are also used. To justify an increase in computational and/or storage requirements, it is necessary to show a corresponding improved efficiency in the solution process.

4.6.1 Aggregating Constraints As a Method for Solving Zero-one Problems :

The ultimate in surrogate constraints, if achievable, is a single constraint which has the same feasible integer solution set as the original set of constraints. Such a surrogate or aggregate constraint can indeed be developed, and in this section we shall consider methods for constructing such aggregate constraints. Once such a constraint has been developed, we merely need to solve an integer programming problem having a single

constraint - a Knapsack problem.

To form the aggregate constraint, we generate a linear combination of the original problem constraints. The procedure to be developed combines two constraints into one, and then combines the resulting constraint with a third constraint, and so on, until a single constraint is achieved.

The Elmaghraby and Wig [1970] method requires that all coefficients appearing in either constraint (being combined) be strictly positive. This can be achieved in general by first making all coefficients non-negative by replacing any x_j having a negative coefficient by $1 - \bar{x}_j$, where $\bar{x}_j$ is the complement of x_j. Then, to make the coefficients strictly positive, add one constraint to the second to form a new constraint. The two new constraints have all coefficients strictly positive, and are equivalent to the original constraints. (This construction of constraints is a method developed by Mathews [1897].)
Designating the new constraint as

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad , \qquad i = 1, 2$$

We now wish to find multipliers for the constraints for aggregation purposes. Let l be the multiplier for the first constraint

Let λ be the multiplier for the second constraint where λ is an integer satisfying expression (4.9)

$$\lambda > b_2 \max_j \left\{ \frac{a_{1j}}{a_{2j}} \right\} \qquad \ldots(4.9)$$

The composite constraint is then

$$\sum_{j=1}^{n} (a_{1j} + \lambda a_{2j}) x_j = b_1 + \lambda b_2$$

The process is repeated using the composite constraint, and so on, until all constraints have been aggregated.

4.6.2 <u>An Improved Method For Generating a Composite Constraint</u> :

A number of improvements have been proposed. Glover and Woolsey [1970] discovered a superior method for combining equations into a composite constraint having smaller coefficients than that generated by the Elmaghraby and Wig method. Kendall and Zionts [1972] have improved upon this method by further reducing the size of the coefficients and the number of variables which, in turn, reduces the computer storage requirements. We describe one of the methods proposed by Kendall and Zionts, which has the advantage of there being no restrictions on the signs of

the entries in the constraints. This implies that no complements of variables are required, thereby economising on computer storage during the aggregation (and possibly solution) process.

The framework of this method is similar to that of the Elmaghraby - Wig method. Two constraints are combined, the result is combined with a third constraint, and so on. However, neither of the two multipliers is permitted to be one.

The following theorem for combining two constraints is the basis of the method.

4.6.3 Theorem : Given two constraints of the following form :

$$\sum_{j=1}^{n} a_{1j} x_j = b_1 \qquad \ldots(4.10)$$

$$\sum_{j=1}^{n} a_{2j} x_j = b_2 \qquad \ldots(4.11)$$

$$x_j \geq 0 \text{ and integer}$$

$$x_1, \ldots, x_p \leq 1 \quad (p \leq n)$$

Then equations (4.10) and (4.11) are equivalent to the equation (4.12)

$$\lambda_1 \sum_{j=1}^{n} a_{1j} x_j + \lambda_2 \sum_{j=1}^{n} a_{2j} x_j = \lambda_1 b_1 + \lambda_2 b_2 \qquad \ldots(4.12)$$

Where the multipliers $\lambda_i > 0$ satisfy the following conditions:

1. $$\lambda_1 > b_2 - \sum_{j=1}^{n} a_{2j}^- - \min_{a_{2j} \neq 0} \{| a_{2j} |\}$$

$$\lambda_2 > b_1 - \sum_{j=1}^{n} a_{1j}^- - \min_{a_{1j} \neq 0} \{|a_{1j}|\}$$

2. $$\frac{b_2 - \sum_{j=1}^{n} a_{2j}^-}{\lambda_1} \quad \text{and} \quad \frac{b_1 - \sum_{j=1}^{n} a_{1j}^-}{\lambda_2}$$

are not integers.

3. λ_1 and λ_2 are relatively prime. (The greatest common divisor of λ_1 and λ_2 is one.)

Proof : Equations (4.10) and (4.11) clearly simply (4.12). To prove that equation (4.12) implies equations (4.10) and (4.11), we note that

$$\sum_{j=1}^{n} a_{1j} x_j = b_1 - \frac{(\sum_{j=1}^{n} a_{2j}x_j - b_2)\lambda_2}{\lambda_1}$$

$\sum_{j=1}^{n} a_{1j} x_j$ and b_1 are both integer, hence $(\sum_{j=1}^{n} a_{2j} x_j - b_2) \lambda_2/\lambda_1$ is integer. From the condition that λ_1 and λ_2 must be relatively prime, we see that $(\sum_{j=1}^{n} a_{2j} x_j - b_2)/\lambda_1$ must be integer. Using

expression (4.12) we have

$$\frac{\sum_{j=1}^{n} a_{2j} x_j - b_2}{\lambda_1} = \frac{b_1 - \sum_{j=1}^{n} a_{1j} x_j}{\lambda_2} = q$$

where q is an integer.

We now prove q is zero by contradiction. First we assume $q > 0$. This implies that $(b_1 - \sum_{j=1}^{n} a_{1j} x_j)/\lambda_2$ is a positive integer. We may now write

$$\frac{b_1 - \sum_{j=1}^{n} \bar{a}_{1j} + \sum_{j=1}^{n} \bar{a}_{1j} - \sum_{j=1}^{n} a_{1j} x_j}{\lambda_2} = q \geq 1 \quad \ldots(4.13)$$

Since $(b_1 - \sum_{j=1}^{n} \bar{a}_{1j})/\lambda_2$ is not integer (because of condition 2) and hence not zero, neither is

$$\frac{\sum_{j=1}^{n} \bar{a}_{1j} - \sum_{j=1}^{n} a_{1j} x_j}{\lambda_2}$$

By noting that

$$\sum_{j=1}^{n} \bar{a}_{1j} x_j - \sum_{j=1}^{n} a_{1j} x_j = \sum_{j=1}^{n} \bar{a}_{1j} (1-x_j) - \sum_{j=1}^{n} a_{1j} x_j \quad \ldots(4.14)$$

Substituting expression (4.14) into (4.13) gives

$$b_1 - \sum_{j=1}^{n} \bar{a}_{1j} - \min_{a_{1j} \neq 0} \{| a_{1j} |\} \geq \lambda_2$$

a contradiction to condition 1. A very similar argument shows that q cannot be negative. Hence, q must be zero, in which case both constraints (4.10) and (4.11) are satisfied. This completes the proof.

In actual practice, the smallest value of λ's that satisfy the above criteria are chosen in order to help keep the magnitude of coefficients in the composite constraint small.

4.7 OTHER DEVELOPMENTS IN AGGREGATING CONSTRAINTS

Other methods for aggregating constraints have been proposed by Glover and Woolsey [1970] and Bradley [1971,1970]. Some allow for considerable flexibility in choosing multipliers, but systematic comparison of the methods has yet to be made. It is clear that the order in which constraints are combined makes a difference. Thus, assuming smaller

coefficients in the aggregate constraint are desirable, further study of the order of the constraints in the aggregation process is appropriate.

The methods described in this section are zero-one variables. However, they may be extended to bounded integer variables, so the method is applicable to general all-integer problems.

4.8 ADVANTAGES AND DISADVANTAGES OF AGGREGATING SCHEMES

All of the aggregation schemes have the same disadvantage- the coefficients become too large to be stored in one computer word as an integer. Since it is necessary to maintain accuracy in an integer programming problem, a multiple precision package of subroutines (subroutines that store the integer in more than one computer word and execute the basic operations of add, subtract, and multiply in that manner) must be utilized. This set of subroutines would be the major part of a computer program using an aggregation approach.

One advantage of these methods is the simplicity of solving the single constraint problem that is generated once the constraints are aggregate , even though solving the resulting problem

is more difficult than solving a Knapsack problem having an inequality constraint. A second major advantage derives from the recursive nature of the algorithms. It is necessary to store only two constraints in computer memory. Further, the objective function is not considered until all the constraints are combined into one.

The possibility of writing an efficient computer program using a constraint aggregation algorithm and a multiprecision package of subroutines is not farfetched. Even though multiprecision required more than one computer word per coefficient, the storage requirements for the coefficients of the aggregated constraint will be much less than the storage requirements for all of those constraint of the original problem. This implies that such a program would be particularly valuable to a user restricted by the amount of available core storage. This method would alow a person using a time-sharing (and consequently "coresharing") system or a limited memory minicomputer to solve reasonable large problems which would be impossible to solve using presently available integer programming codes. If appears that constraint aggregation methods warrant serious study as a feasible alternative for solving integer programming problems.

CHAPTER-V

ALGORITHMS FOR SPECIALIZED INTEGER MODELS

5.1 INTRODUCTION

This chapter presents algorithms that exploit the special properties of certain integer models, including the Knapsack, the fixed charge, the travelling salesman, and the set covering problems. Although the general algorithm presented in the proceding chapters can be used, at least in principle, to solve the indicated models, the specialized methods should prove efficient computationally.

5.2 THE KNAPSACK PROBLEM

Consider the situation in which a hiker must decide on the various items he could take under the limitation, that he cannot carry more than a specified weight. His objective is to maximize the total value of the items he takes. This situation can be represented by the following:integer program with a single constraint :

$$\text{maximize} \quad z = \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j \geq 0 \text{ and inteher } (j=1,2,\ldots,n)$$

where the costs c_j, coefficients a_j, and right hand side (number) b are integers, and each a_j (j=1,2,...,n) and be are positive. Such a program is called a Knapsack problem. The name is in reference to packing aKKnapsack which can hold a maximum weight b. Each of n items from which the selection is made has a weight a_j and relative value c_j. The problem is to fill the Knapsack so that its capacity is not exceeded and the total value - that is, the sum of all relative values - is maximized. The model is precisely the above problem where each variable x_j is either 0 (item j is not packed) or 1 (item j is packed).

The above formulation is referred to as a one-dimensional Knapsack problem. A multi-dimensional Knapsack problem include more than one constraint. For example, a volume restriction may also be imposed in addition to the weight constraint.

Although the simplest integer program, the Knapsack problem is worth studying because : (a) it is representative of many industrial situations such as capital budgeting, project selection and capital investment, budget control, and numerous loading problems, and (b) it appears as a subproblem that has to be solved in many integer programming algorithm.

Algorithms which solve the Knapsack problem are usually of the dynamic programming, enumerative, Lagrangian multiplier,or network type.

Early papers which discuss the Knapsack problem in context of dynamic programming include Bellman [1954], [1956] , [1957], [1957], and Dantzig [1957]. Later expositions are in Bellman and Dreyfus [1962] and Dantzig [1963]. Computational improvements in dynamic programming algorithms are described by Greenberg [1969], Dreyfus and Prather [1970], and Yormark [1972].

Enumerative algorithms specialized to solve the Knapsack problem include the branch and bound procedure for zero-one problem appearing in Kolesar [1967], and Faaland [1973]. Everett [1963] introduced the concept of utilizing Lagrangian multipliers to solve discrete programming problems. The Knapsack problem can also be represented by a network in which a minimal cost or shortest route is sought. Shapiro [1968], [1971] discusses the formulation and solution techniques.

The applications involving the Knapsack problem or its solution include the cutting stock problem extensively discussed by Gilmore and Gomory [1961], [1963], [1965], [1966], the journal

selection problem appearing in Kraft and Hill [1973], [1971] and in Glover and Klingman [1973] and the capital investment problem as explained by Hansmann [1961].

Using a basic number theoretic result appearing in Mathews [1897], it can be shown that an integer program with any finite number of constraints can be transformed to Knapsack problem. Although the transformed (Knapsack) problem has the same number of variables, its constraint coefficient are usually enormous. Thus, roundoff errors and overflow often make computer solution impossible. An excellent survey regarding transformation is in Kendall and Zionts [1972].

Computational results with Knapsack agorithms are in Cabot [1970], Cabot and Hurter [1968] , Gilmore and Gomory [1963], Guignard and Spielberg [1971], Gulleu, Swanson, and Woolsey| |, Kolesar [1967], and in Weingarther and Niss [1967]. A generall survey article is in Salkin and Dekluyver [1972].

Recently the work is done on the multiple choice Knapsack problem. The problem is discussed in Ibaraki, Hasegawa, Teranak and Iwase [1978], and Sinha and Soltners [1979]. The linear multiple choice Knapsack problem is in Zemel [1980]. Algorithm

for multiple choice Knapsack linear program is in Dyer [1984]. Continuous maximum Knapsack problems are discussed in Eiselt [1986]. Dudzinski, Krzysztof, Walukiewies, Stanislaw [1987] discusses the wellknown Knapsack problem and its two generalisations. The first generalisation is in the form of the multiple choice Knapsack problem and the second is in the form of the nested Knapsack problem.

5.2.1 Applications and Uses of the Knapsack Problem :

(a) Capital Budgeting :

In the simplest case the problem consists of choosing among n competing independent investment possibilities so as to maximise the total payoff from the investment subject to a constraint on available funds. To model the problem, let c_j ($j = 1,\ldots,n$) be the payoff from including project j, a_j($j=1,..,n$) be the cost of project j, and b be the total available funds. Further, let x_j be o(1) if project is rejected (accepted). The basic model is then

$$\text{maximise} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_j x_j \leq b$$

$$\text{and} \quad x_j = o \quad \text{or} \quad 1 \qquad (j = 1,\ldots, n)$$

which is a Knapsack problem with zero-one variables. Notice that the parameters c_j, a_j and b may be taken as positive and integral.

The investment situation just described often can be divided into several periods with costs and available funds broken down to period requirements. In this situation, the non-negative cost of project j in period t is denoted by a_{tj} $(j=1,\ldots,n, t=1,\ldots,T)$ and the available funds in period t is b_t $(t = 1,\ldots, T)$. The multi-period Lorie-Savage capital budgeting model is then

$$\text{maximize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_t \qquad (t=1,\ldots,T)$$

$$\text{and} \quad x_j = 0 \text{ or } 1 \qquad (j=1,\ldots,n)$$

Unless $T = 1$, the above problem is a standard zero-one integer program with a non-negative T by n constraint matrix (project j may not require any funds in period t, so it is possible to have $a_{tj} = 0$. We may also note that an integer program with more than one constraint and a non-negative coefficient matrix is some times termed as a multidimensional Knapsack problem), and may be solved by the search enumeration techniques discussed

in Chapter-IV. Other solution possibilities include aggregating the T constraints into a single constraint with the same set of integer solutions and solving the resulting Knapsack problem, using a Lagrangian multiplier approach to find a near optimal solution, or possibly dynamic programming when there are only a few periods.

A second extension to the model occurs when the n investment possibilities are partitioned into disjoint subsets $n_1, \ldots, n_p$ ($p \leq n$) and it is specified that precisely one project in each subset must be selected. The additional requirements have the form

$$\sum_{j \in n_i} x_j = 1 \qquad (i=1, \ldots, p \leq n)$$

Equalities of this type are called multiple choice constraints, since each one specifies (when x_j = or or 1) that a single variable is selected to be set to 1 among a set of n_i variables.

An algorithm for an integer program as of the one appearing in (5.1) should exploit the structure of the multiple choice constraints. For example, they dictate obvious branching rules in an enumeration and natural constraint feasibility tests,

and they can be handled implicitly using a technique called generalized upper bounding (Lasdon [1970]) when linear programming is used.

(b) The Cutting Stock Problem :

There are numerous situations in which a material comes in standard lengths and must be cut up according to requirements and costs (or profit). These include goods which come up in rolls (e.g. paper, sheet metals, and textiles) as well as less flexible materials (e.g. glass, metal, pipe, and other tubing).

(i) The One Dimensional Case :

Consider the case in which only straight vertical cuts are allowed. That is, each standard length or roll is to be sliced into lengths l_i $(i=1,\ldots,m)$. There is no limitation on the number of pieces avalable of each length. All stocked pieces have the same width. The cutting stock problem is to cut up rolls of the material so that the demand for the number of pieces of each length $l_i (i=1,\ldots,m)$ is satisfied while the total cost of the rolls used is minimized.

The problem can be expressed as an integer program by letting N_j $(j=1,\ldots m)$, to be number of pieces of length l_i

demanded, x_j (j=1,..., n), the number of times the jth cutting pattern is used (each time to cut up a roll), c_j, the cost of the roll from which jth cutting pattern is cut, and a_{ij}, the number of pieces of length ℓ_i produced each time the jth cutting pattern is used. The cutting stock problem is then

$$\text{minimize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \geq N_i \qquad (i=1,\ldots,m)$$

$$x_j \geq 0 \text{ and integer} \quad (j=1,\ldots,n)$$

The obvious difficulty with this formulation is that x_j is restricted to integer values, so that for a large number of variables the problem may not be computationally tractable. Another difficulty is that it may be impossible to enumerate all the cutting patterns in advance.

Gilmore and Gomory account for these difficulties as follows :

First, drop the integer condition on x_j so that the resulting problem becomes a regular linear program, then round the optimal continuous solution in some appropriate manner. Second, instead of enumerating all cutting patterns in advance, a pattern is

considered only if it is promising from the view point of improving the linear programming solution. This is where the Knapsack model proves valuable.

The linear programming procedure developed by Gilmore and Gomory [1961], [1963] is a standard revised simplex algorithm in which the column that enters the basis is found by solving one or more Knapsack problems. Computational experiences indicates that a specialized enumeration algorithm appears to be the best way of solving the Knapsack problems.

(ii) The Two-Dimensional Case :

Suppose now that we have the same problem as before except that straight width cuts are allowed and N_i is the demand for the number of rectangles of length $\quad_i$ and width w_i. However, because of the added dimension the column (cutting pattern) generation scheme can not longer be used. Without explicitly enumerating all the columns, the linear program (hence the integer program) can not be solved.

A related problem is to ignore the demand constraints and find cutting patterns which maximize the value of a roll when it is cut up into rectangles of given sizes. In particular,

we wish to

$$\text{maximize} \quad f(L,W) = p_1 a_1 + \ldots + p_m a_m ,$$

where $a_1, \ldots, a_m$ are nonnegative integers such that there exists a way of dividing a rectangle (roll) (L,W) into a_i rectangles (ℓ_i, w_i) for $i = 1, \ldots, m$ and each rectangle a_i has a value p_i $(i=1,\ldots,m)$. The case where the cuts are always continued to the end of the existing roll (termed "guillotine cuts") is extensively discussed by Gilmore and Gomory [1965], [1966]. Algorithms for this problem which take advantage of certain properties of function $f(L,W)$ are also presented.

Remarks :

The function

$$f(x) = \max\left\{ \sum_{j=1}^{n} c_j x_j \Big/ \sum_{j=1}^{n} a_j x_j \le b_j, x_j \ge 0 \text{ and integer } (j=1,\ldots,n) \right\}$$

is some times termed a one-dimensional Knapsack function, $f(L,W)$ is termed a two dimensional Knapsack function. Properties of one-and-two-dimensional functions are discussed in Gilmore and Gomory [1965], [1966]. Also an enumerative procedure for the integer program representing the one-dimensional cutting stock problem is presented by Pierce [1966], and a heuristic

technique for the two-dimensional case is given by Art [1966].

5.2.2 Reduction of Integer Problems to Knapsack Models :
(Aggregating Constraints)

A system of linear equation with integer coefficients can usually be transformed to a single linear equation which has the same set of non-negative integer solutions as the parent equations. This means that the constraints of an integer program can first be transformed to a single constraint and the integer program can then be solved by solving the Knapsack problem. If the resulting Knapsack problem is easier to solve than the integer program and its construction can be accomplished in a reasonable amount of time, the transformation is worthwhile.

Consider the m linear equations

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad (i=1,\ldots,m) \qquad \ldots(5.1)$$

with every a_{ij} and b_i integer. The problem is to find weights $w_1,\ldots,w_m$ so that every non-negative integer solution to the single equation

$$\sum_{j=1}^{n} a_j x_j = b \qquad \ldots(5.2)$$

where $a_j = \sum_{i=1}^{m} w_i a_{ij}$ $(j=1,\ldots,n)$ and $b = \sum_{i=1}^{m} w_i b_i$, is a solution

to (5.3). Observe that, since the constraints (5.1) imply (5.2), every non-negative integer solution to (5.1) is a solution to (5.2). For arbitrary weights, however, the set of non-negative integer $x = (x_1,\ldots,x_n)$ satisfying (5.2) is usually larger than the set satisfying (5.1).

(i) An Aggregation Process :

A theorem originating in Mathews [1897] indicates how to aggregate two equations with positive coefficients so that the non-negative integer set does not enlarge. This theorem enables us in reducing multiple constraints problems into a single constraint.

Theorem : (Mathews [1897])

Consider a system of two linear equations

$$s_1 = \sum_{j=1}^{n} a_{1j} x_j = b_1 \qquad \ldots(5.3)$$

$$s_2 = \sum_{j=1}^{n} a_{2j} x_j = b_2 \qquad \ldots(5.4)$$

with strictly positive integer coefficients a_{ij} ($i=1,2$, and $j=1,\ldots,n$)

(a) If there exist non-negative values $x_1,\ldots,x_n$ satisfying (5.3) and (5.4), then

$$b_2 a_{1j} / a_{2j} \geq b_1 \quad \text{for atleast one } j \quad (1 \leq j \leq n),$$

(b) If w is any positive integer such that

$$w > b_2 \max_j \left\{ a_{1j} / a_{2j} \right\}$$

then the solution set of (5.3) and (5.4) in non-negative integer variables is the same as that of the single equation

$$s_1 + ws_2 = b_1 + wb_2 \qquad \ldots(5.5)$$

Proof: (a) Suppose $b_2 a_{1j}/a_{2j} < b_1$ for every j. As $a_{1j} > 0$ and $a_{2j} > 0$, for $x_j \geq 0$ it follows that $b_2 a_{1j} x_j < b_1 a_{2j} x_j$. Summing over j yields

$$b_2 \sum_{j=1}^{n} a_{1j} x_j < b_1 \sum_{j=1}^{n} a_{2j} x_j$$

for every $x_j \geq 0$. But, by hypothesis, there is at least one non-negative solution x_j^0 to (5.3) and (5.4), so that

$$\sum_{j=1}^{n} a_{1j} x_j^0 = b_1 \quad \text{and} \quad \sum_{j=1}^{n} a_{2j} x_j^0 = b_2$$

Substituting in the last inequality produces $b_2 b_1 < b_1 b_2$, which is a contradiction.

(b) For any solution to (5.3) and (5.4), we have $s_1 = b_1$ and

$s_2 = b_2$, which means that $ws_2 = wb_2$ and $s_1+ws_2 = b_1+wb_2$, or it is a solution to (5.5). To prove the converse, consider any non-negative integer solution x_j^o to (5.5). If it is not an integer solution to (5.3) and (5.4), then $s_2 \neq b_2$, because if $s_2 = b_2$, it follows that $ws_2 = wb_2$, and subtracting in (5.5) yields $s_1 = b_1$. If

$$s_2 = \sum_{j=1}^{n} a_{2j} x_j^o \neq b_2,$$

then there is a non-zero integer q such that $s_2 = b_2 + q$. Substituting $b_2 + q$ for s_2 in (5.5) (with $x_j = x^o{}_j$) yields $s_1 + w(b_2+q) = b_1 + wb_2$ or $s_1 = b_1 - wq$. We show that q must be zero. Since the coefficient a_{1j} are positive,

$$s_1 = b_1 - wq \geq o$$

for any non-negative (integer) solution.

For $q > o$, $b_1 - wq \geq o$ implies $b_1 \geq wq > b_1 q$, or $1 > q$, which is a contradiction where $w > b_1$ follows from the hypothesis and (a), and is used to obtain $wq > b_1 q$.

When $q < o$, it must be $-1, -2, -3$ etc. and thus $s_1 = b_1 - wq$ implies $s_1 \geq b_1 + w$. But, by definition,

$$w > b_2 \text{ maximum } \frac{a_{1j}}{a_{2j}}$$

and thus $wa_{2j} > b_2a_{1j}$ for $j = 1,\ldots,n$. Multiplying both sides by x_j and summing yields $ws_2 > b_2s_1$. Since $b_1b_2 \geq 0$, this implies $ws_2 > b_2s_1 - b_1b_2$. Substituting $s_1 \geq b_1 + w$ gives $ws_2 > wb_2w$, or $s_2 > b_2$. But $s_2 = b_2 + q$ and $q < 0$, so $s_2 < b_2$, hence a contradiction. So q must be 0 and the theorem is proved.

Mathews observed that this theorem can be applied to two equations with non-negative (but not necessarily all positive) integer coefficients by first replacing them with the pair

$$s_1 + s_2 = b_1 + b_2 \qquad \ldots(5.6)$$

$$s_1 + 2s_2 = b_1 + 2b_2 \qquad \ldots(5.7)$$

Equations (5.6) and (5.7) have strictly positive coefficients since they are of the form $a_{1j}+a_{2j}$ (in (5.6)), or $a_{1j}+2a_{2j}$ (in (5.7)), and for each j ($j=1,\ldots,n$), $a_{ij} > 0$ for at least one i ($i = 1,2$). Thus, these equations can assume the role of (5.3) and (5.4) in Mathews' theorem. Also, note that any solution to (5.6) and (5.7) is a solution to (5.3) and (5.4) (and conversely). To see this, just subtract (5.6) from (5.7)

to obtain $s_2 = b_2$ and then, by (5.6), $s_1 = b_1$.

Elmaghraby and Wig [1970] have further noted that Methews' theorem can be used to aggregate a system of equations (5.1) with non-negative coefficients by recursively using the construction (5.6) and (5.7). That is, the first two equations in the system are replaced by (5.6) and (5.7) and then aggregated. The aggregated equation becomes the first equation and replaces the first two, so that the system now has one less equation. The process is then repeated unitl a single equation is left.

We can also use the above pairwise aggregation process on systems of equations containing negative coefficients so long as upper bounds on the corresponding variables can be found. In particular, if $a_{1j} < o$ and it is known that $x_j \leq u_j$ (some positive integral upper bound), then substituting $u_j - \bar{x}_j$ for x_j is constraint 1, where the complementing variable $\bar{x} = u_j - x_j$, changes the sign of a_{1j}. Notice that this and the requirements in Methews' theorem mean that any integer program which has a bounded linear programming feasible region with at least one integer point can be transformed to an equivalent Knapsack problem.

(ii) An Improved Aggregation Process (Glover [1972])

Theorem : Consider a system of two equations

$$s_1 = \sum_{j=1}^{n} a_{1j} x_j = b_1 \qquad \ldots(5.8)$$

$$s_2 = \sum_{j=1}^{n} a_{2j} x_j = b_2 \qquad \ldots(5.9)$$

where all coefficients (a_{ij}, b_i) are integers, and at least one of b_1 and b_2 is not zero. Let w_1 and w_2 be relatively prime (nonzero) integers (their greatest common divisor is plus or minus 1). If there exists at least one non-negative integer soltution to (i) and (ii), then every non-negative integer solution to

$$w_1 s_1 + w_2 s_2 = w_1 b_1 + w_2 b_2 \qquad \ldots(5.10)$$

is a non-negative integer solution to (5.8)and (5.9) , and conversely, provided that

$$w_1 a_{1j} + w_2 a_{2j} \geq | b_2 a_{1j} - b_1 a_{2j} | \qquad \ldots(5.11)$$

for $j = 1,\ldots,n$ and (5.11) holds as a strict inequality for j in J, where J is any non-empty subset of $\{1,\ldots,n\}$ such that all non-negative solutions to(5.10) satisfy $x_j > o$ for

atleast one j in J.

Proof: Clearly (5.8) and (5.9) imply (5.10), so that every (non-negative integer) solution to (5.8) and (5.9) satisfies (5.10). To show the converse, select any non-negative integer solution x_j $(j=1,\ldots,n)$ to (5.10) and multiply both sides of (5.11) by $x_j = |x_j|$, so that

$$(w_1a_{1j}+w_2a_{2j})x_j \geq |b_2a_{1j}-b_1a_{2j}|\,|x_j| = |(b_2a_{1j}-b_1a_{2j})x_j|$$

Summing the last expression over j gives

$$w_1b_1 + w_2b_2 > |\,b_2s_1 - b_1s_2\,|, \qquad \ldots(5.12)$$

where the strict inequality follows from the definition of J. Now suppose $s_1 \neq b_1$, which implies $s_2 \neq b_2$ (and vice-versa). Then let α and β be any integers such that

$$s_1 = b_1 + \alpha \quad \text{and} \quad s_2 = b_2 - \beta$$

For the current integer solution, $w_1s_1 + w_2s_2 = w_1b_1+w_2b_2$. Substituting for s_1 and s_2 yields

$$w_1b_1 + w_1\alpha + w_2b_2 - w_2\beta = w_1b_1 + w_2b_2,$$

thus, $w_1\alpha = w_2\beta$. But , by hypothesis, w_1 and w_2 are relatively

prime. Hence, for the last equality to be true $\alpha = qw_2$ and $\beta = qw_1$, where q is an integer. This follows because $\beta=(w_1/w_2)\alpha$, and w_1/w_2 is not an integer, so only way for β to be an integer is for α to be a multiple of w_2. A similar argument can be made for β. Substituting for α and β in expressions for s_1 and s_2 gives $s_1 = b_1 + qw_2$ and $s_2 = b_2 - qw_1$. Rgese last two inequalities in (5.2) give

$$w_1b_1 + w_2b_2 > |b_2b_1 + b_2qw_2 - b_1b_2 + b_1qw_1| = |q|\,|w_1b_1 + w_2b_2|,$$

which means that $|q| = 0$ or $q = 0$, and thus $s_1 = b_1$ and $s_2 = b_2$, completing the proof.

The theorem implies by (5.12) that w_1 and w_2 be chosen so that $w_1b_1 + w_2b_2 > 0$. Alsom by (5.11), the coefficients in (5.10) (i.e. $w_1a_{1j} + w_2a_{2j}$ for $j = 1,\ldots, n$) are nonnegative. Therefore, every non-negative integer solution to (5.10) must have $x_j > 0$ for at least one j where its coefficient in (5.10), $w_1a_{1j} + w_2a_{2j}$, is positive. Consequently, the set J can consist of those j for which $w_1a_1 + w_2a_{2j} > 0$. Condition (5.11) also imposes some additional restrictions on the coefficients in (5.8) and (5.9). By (5.11) a sufficient, but

not necessary, condition for the existence of relatively prime weights w_1 and w_2 is that coefficients a_{ij} are non-negative, since we can then take $w_1 = 1$. This may not yield the aggregated equation with coefficients of smallest magnitude, although it does simplify the computations.

5.2.3 Algorithms for Solving the Knapsack Problem.

An algorithm for solving the Knapsack problem can usually be classified as a dynamic programming technique, a branch and bound enumeration, a Lagrangian multiplier method, or a network method.

Dynamic programming techniques can be found in Belman [1956] and Dantzig [1957]. Branch and bound algorithm for solving the Knapsack problem is in Greenberg and Hegerich [1970], and Kolesar [1967]. Lagrangian multiplier methods for solving the Knapsack problem are discussed in Everett [1963], and Brooks and Geofrion [1966]. Network approaches for solving a Knapsack problem are in Shapiro [1968], [1971].

5.3 THE FIXED CHARGE PROBLEM

The capacitated plant location problem is the mixed integer program :

$$\text{(FCP)} \qquad \text{minimise} \sum_{i=1}^{m} \sum_{j=1}^{n} g_{ij} s_{ij} + \sum_{i=1}^{m} f_i x_i = z \qquad \ldots(5.13)$$

$$\text{subject to} \sum_{i=1}^{m} s_{ij} = d_j \qquad (j=1,\ldots,n) \qquad \ldots(5.14)$$

$$\sum_{j=1}^{n} s_{ij} \leq M_i x_i \qquad (i=1,\ldots,m) \qquad \ldots(5.15)$$

$$s_{ij} \geq 0 \qquad (\text{all } i,j) \qquad \ldots(5.16)$$

$$\text{and} \qquad x_i = 0 \text{ or } 1 \qquad (i=1,\ldots,m) \qquad \ldots(5.17)$$

where the model represents the situation in which there are m plants that produce a single commodity for n customers. Each plant i can produce at most M_i units and each customer j requires d_j units, f_i is a positive fixed cost associated with plant i, and g_{ij} is a positive per unit shipping cost from plant i to customer j. The variables are s_{ij} and x_i, which represent the amount shipped from i to j and whether a plant is open ($x_i = 1$) or closed ($x_i = 0$) respectively. The structure of the problem suggests several useful results.

In 1954 Hirsch and Dantzig formulated the general fixed charge problem. Other formulations include the fixed cost transportation problem by Balinski [1961], and the plant location problem by Balinski [1964] and by Efroymson and Ray [1966].

Algorithms for those and related problems include the enumerative techniques appearing in Brown [1973], David and Ray [1969], Efroymson and Ray [1966], Gray [1967], [1977], Jones and Soland [1969], Khumawala [1972], Marsten [1972], Pinkus, Gross, and Soland [1973], Sa [1969], Spielberg [1964], [1969], [1969], and Spielberg [1970]. A partitioning algorithm is given by Balinski and Wolfe [1963], group theoretic techniques are described by Kennington and Unger [1973], and by Hopkins [1971] and Lagrangian Multiplier approaches are given by Geoffrion [1973]. Heuristic procedures are in Armour and Buffa [1965], Balinski [1961], [1964] Baumol and Wolfe [1968], Cooper [1964], Cooper and Drebes [1967], Densler [1969], Drysdale and Sandiford [1969], Dwyer [1966], Feldman, Lehrer, and Ray [1966], Kuehn and Hamburger [1963], Kuhn and Baumol [1962], Manne [1964], Sa [1969], Shannon and Ignisio [1970], and Steinberg [1970].

Recently, Ross and Soland [1975] presented a branch and bound algorithm for the generalized assignment problem which is a computational study of the fixed charge problem. A computational study of the fixed charge transportation problem with a branch and bound code is given in Kennington [1976]. Related work includes

Kennington and Unger [1976], Waker [1976], Fisk and McKeown [1979], McKeown [1981] and Suhl [1985].

5.3.1 Algorithms for Solving Fixed Charge and Plant Location Problems :

(a) A Branch and Bound Algorithm for the Fixed Charge Problem :

One technique for solving Fixed Charge Problem is by a straightforward branch and bound enumeration. Since the linear program allow the free x_i variables to vary between 0 and 1 they are not transportation problems, and thus they have to be solved by a standard simplex method. If optimal tableau are kept, successive problems may be solved by simple post optimality procedures. In selecting the dangling node to create nodes from, two obvious strategies are (i) pick the dangling node with the lowest linear programming solution, and (ii) select the most recently created dangling node with the lowest linear programming solution. Criterion (i) is intended to obtain the optimal mixed integer solution as quickly as possible; criterion (ii) allows for a simpler bookkeeping scheme to keep track of the enumeration and usually required less computer storage.

Once the dangling node has been selected, a variable which

is at a fractional value in its linear programming solution must be selected to set 0 and 1 so that two new nodes are created. One rule is to select the eligible variable x_i whose plant has the largest capacity M_i. The intent is to obtain a mixed integer solution quickly. As the fixed cost for hight capacity plants may be very large, another criterion is to pick x_i so that it yields the smallest value of f_i/M_i; this ratio may be thought of as the per unit production cost at plant i. Note that criteria involving the capacity M_i lose meaning when solving the plant location problem.

(b) Heuristics for Obtaining Mixed Integer Solutions :

If a good solution to the mixed integer program can be found early, it will natrually curtail the length of the enumeration. Note that when the x_i variables which are at fractional values in the linear programming solution are rounded to 1, we have a solution to the fixed charge (or plant location) problem since the right hand side in the constraint (5.15) increases. Also, observe that an integer solution to problem FCP exists whenever a subset $S \subseteq \{1,\ldots,m\}$ of the plants are open (that is, $x_i = 1$ for $i \in S$) so that their total

capacity exceeds the total demand , (This assumes that every open plant can service all the customers. If this is not the case, more plants than those in S may have to be opened) that is, $\sum_{i \in S} M_i \geq \sum_{j=1}^{n} d_j$. Thus one way to obtain a solution is to open a minimum number of plants (S) so that all the demands can be met and the total fixed cost (for the total value of f_i/M_i) for the plant opened is minimised. The values of the shipping variables z_{ij} can then be found by solving the resulting transportation problem (where $x_i = 1$ for $i \in S$ and $x_i = 0$ for $i \notin S$).

(c) A Branch and Bound Algorithm for the Plant Location Problem: (Efroymson and Ray [1966])

The efficiency of a branch and bound algorithm largely depends on how quickly the linear programs can be solved. When each plant can satisfy all customer demands, it is possible to reformulate the plant location problem (5.13) through (5.17) so that the linear programs can be solved by inspection. To transform the problem, define y_{ij} to be the fraction of customer j's demand satisfied by plant i, that is

$$y_{ij} = z_{ij} / d_j \qquad \ldots(5.18)$$

for all i,j. Before making direct use of (5.16), we first note that when $M_i \geq \sum_{j=1}^{n} d_j$ $(i=1,\ldots,m)$, the inequality (5.15) is only necessary to ensure that shipments are not allowed from a closed plant. This allows us to replace constraint (5.15) by

$$\sum_{j=1}^{n} y_{ij} \leq n x_i \qquad (i=1,\ldots,m) \qquad \ldots(5.15)'$$

since $x_i = 0$ implies $y_{ij}=0$ $(i=1,\ldots,m)$ and thus no demand will be satisfied by plant i. Now, substituting $y_{ij} d_j$ for z_{ij} in expressions (5.13), (5.14) and (5.16), and letting $c_{ij}=g_{ij}d_j$ yields the equivalent plant location problem

(PLP) minimise $$\sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij} y_{ij} + \sum_{i=1}^{m} f_i x_i \qquad \ldots(5.13)'$$

subject to $$\sum_{i=1}^{m} y_{ij} = 1 \quad (j=1,\ldots,n) \qquad \ldots(5.14)'$$

$$\sum_{j=1}^{n} y_{ij} \leq n x_i \quad (i=1,\ldots,m) \qquad \ldots(5.15)'$$

$$y_{ij} \geq 0 \qquad (\text{all } i,j) \qquad \ldots(5.16)'$$

and $$x_i = 0 \text{ or } 1 \quad (i=1,\ldots,m) \qquad \ldots(5.17)'$$

Problem PLP with $x_i = 0$ or 1 replaced by $0 \leq x_i \leq 1$ $(i=1,\ldots,m)$ may be solved by inspection. In particular, every optimal linear programming solution will satisfy the constraints (5.14)', and

as $f_i > 0$, will have the smallest non-negative values for x_i ($i=1,\ldots,m$), so that the constraints (5.15)' are satisfied. Therefore, at any linear programming optimum

$$\sum_{j=1}^{n} y_{ij} = nx_i \qquad (i=1,\ldots,m)$$

$$\text{or} \quad \frac{1}{n} \sum_{j=1}^{n} y_{ij} = x_i$$

Substituting for x_i in the linear program yields

LP^0 minimise $\sum_{i=1}^{m} \sum_{j=1}^{n} (c_{ij} + f_i/n) y_{ij}$ ...(5.13)'

subject to $\sum_{i=1}^{m} y_{ij} = 1 \quad (j=1,\ldots,n)$ ...(5.14)'

and $y_{ij} \geq 0 \quad$ (all i,j) ...(5.16)'

An optimal solution to this problem is

$$y_{ij} = \left\{ \begin{array}{ll} 1 & \text{if } c_{ij}+f_i/n = \text{minimum } (c_{kj}+f_k/n) \\ 0 & \text{otherwise} \end{array} \right\} \quad \ldots(5.19)$$

for $j = 1,\ldots,n$ and thus

$$x_i = \frac{1}{n} \sum_{j=1}^{n} y_{ij} \qquad (i=1,\ldots,m)$$

Expression (5.19) can be generalised so that the linear programs which appear during a branch and bound enumeration may also be solved by inspection. At any node, let I_0, I_1 and I_f be the set of indices i such that x_i is fixed at 0, fixed at 1, and free, respectively. Therefore the linear program LP^0 becomes

$$LP^0 \qquad \sum_{i \in I_1} f_i + \text{minimise} \left(\sum_{i \in I_f} \sum_{j=1}^{n} (c_{ij} + f_i/n) y_{ij} + \sum_{i \in I_1} \sum_{j=1}^{n} c_{ij} y_{ij} \right)$$

$$\text{subject to} \quad \sum_{i \in I, UI_j} y_{ij} = 1 \qquad (j=1,\ldots,n)$$

$$\text{and} \quad y_{ij} \geq 0 \qquad (\text{all } i,j, i \notin I_0)$$

An optimal solution to this problem may be written as

$$y_{ij} = \begin{cases} 1 & \text{if } c_{ij} + (\ell_i/n) = \underset{k \in I_1 UI_f}{\text{minimum}} (c_{kj} + \ell_k/n) \\ 0 & \text{otherwise} \end{cases}$$

for $j=1,\ldots,n$ and ...(5.20)

$$x_i = \frac{1}{n} \sum_{j=1}^{n} y_{ij} \qquad (i \in I_f)$$

where

$$\ell_k = \begin{cases} f_k & \text{if } x_k \text{ is free} \qquad (k \in I_f) \\ 0 & \text{if } x_k \text{ is fixed at 1} \qquad (k \in I_1) \end{cases}$$

Using (5.20), the linear programs are solved by inspection. The other ingredients in the branch and bound enumeration (e.g. node and branch selection) are essentially the same. Notice that solutions to the plant location problem may be found by partitioning the set of plants $\{1,\ldots,m\}$ into the subsets I_1 and I_0 and using (5.20) with I_0 null. It is not clear how to choose I_1 (which must have at least one element), so that this heuristic produces a good solution.

5.3.2 The Uncapacitated Plant Location Problem (A Computational Experience)

Efroymson and Ray [1966] report that their branch and bound algorithm was able to solve a number of 50 plants and 200 customer problems. The average computer time was 10 minutes on IBM 7094. In addition, Jain [1972] has successfully solved several 39-plant and 63-customer problems using Efroymson and Ray's method. Solution time ranged from 1 to 45 minutes on IBM 360/44. A search enumeration technique which has a generalised origin/restart capacity and a very sophisticated point algorithm was developed by Spielberg [1969], [1969] Computational experience is reported on numerous problems ranging in size from

20 plants and 35 customers to 100 plants and 150 customers. Solution time ranging between 1 minute on IBM 360/44 computer for a 20 by 35 problem to 60 minutes on IBM 7094 for a 100 by 150 problem.

Ciochetto, Swanson, Lee and Woolsey [1972] report that their enumerative algorithm solved ten problems whose size ranged from 5 plants and 72 customers to 50 plants and 50 customers. Times, using a PDP 10 computer, ranged between 1.75 seconds for a 10 by 10 problem to 23 seconds for a 20 by 300 problem. The enumerative algorithm developed by Marsten [1972] was applied to ten 100 customer problems with 8 to 16 plants. Except for the116 by 100 problem, which the algorithm failed to solve, the heuristic used to obtain an initial solution produced an average error of 2 % and, on the average, required 2.2 seconds to obtain. The average solution time for the nine solved problems was 12.7 seconds using a CDC 6400 computers.

A branch and bound algorithm for a plant location type problem where the demand constraints (5.14)' are replaced by

$$\sum_{j=1}^{n} a_{ij} y_{ij} \leq b_i x_i \qquad (i=1,\ldots,m) \qquad \ldots(5.21)$$

is given by Brown [1973]. In inequality (5.21), a_{ij} and b_i are non-negative known constants, the y_{ij} are zero-one variables, and the variables x_i may take on any non-negative integer value. Using a UNIVAC 1108 computer, five problems ranging in size from 10 x_i variables and 100 y_{ij} variables to 19 x_i variables and 950 y_{ij} variables were solved in about 36 seconds each. A heuristi used to produce the first solution gave an average error of 12.2 %

5.4 THE SET COVERING PROBLEM

The set covering problem (SC) is the zero-one integer program

$$\text{minimise} \quad cx \tag{}$$

$$\text{subject to} \quad Ex \geq e \tag{5.22}$$

$$\text{and} \quad x_j = 0 \text{ or } 1 \quad (j=1,\ldots,n) \tag{5.23}$$

where $E = (e_{ij})$ is an m by n matrix whose entries e_{ij} are 0 or 1, $c = (c_j)$ $(j=1,\ldots,n)$ is a cost row with positive components, $x = (x_j)$ $(j = 1,\ldots,n)$ is a vector of zero-one variables, and e is an m vector of (5.22)'s. If $Ex \geq e$ constraints are replaced by the equalities

$$Ex = e \tag{5.22}'$$

the integer program is referred to as a set partitioning problem (SP). Note that, in either case, the constraints (5.23) are equivalent to $x \geq 0$ and integer, since in any minimal solution these conditions imply $x_j = 0$ or 1 for $j = 1,\ldots,n$ (This is evident in the set partitioning problem. For the set covering problem, suppose x is optimal and it has a component x_j greater than 1. Then x_j can be reduced to value 1 and the constraints (5.22) are still satisfied, as the costs are positive, we have an improved solution contradicting that x is optimal). Also, if we think of the columns of E and e as sets, the set covering problem is equivalent to finding a cheapest union of sets from E that covers every component of e, where component i of e is covered if at least one of the selected sets (columns) from E has a 1 in row i. In the set partitioning problem, we seek a cheapest union of disjoint sets from E which covers e.

The set covering and the set partitioning problems are the representative of numerous real world situations. In some of the applications, in addition to the constraints (5.22) or (5.22)' and (5.23), a third set of constraints appears. These constraints are often of the form

$$L \leq Dx \leq U, \qquad \ldots(5.24)$$

where D is an m' by n non-negative matrix having row diagonal structure, L is a non-negative m' vector, and U is a positive m' vector. The inequalities (5.24) are sometimes referred to as base constraints and a programming problem with (5.24) or (5.22)' (5.23), and (5.24) as a base constrained set covering (or partitioning) problem. Because of the extensive applicability of the set covering and set partitioning model, considerable effort has been devoted to developing special purpose algorithms which take into account the positive costs, the zero-one constraint matrix, the right-hand side of (5.22).

Certain network problems can be modeled as a set covering or set partitioning problem with a particular E matrix. The proble and formulations are in Balinski [1965], Balinski and Quandt [1964], Bellmore, Greenberg and Jarvis [1970], Bellmore and Ratloff [1971], and Garfinkel and Nemhauser [1972], [1972]. For the set covering and set partitioning problems enumerative techniques are described by Garfinkel and Nemhauser [1969], [1970], Guha [1973], Lemke, Salkin and Spielberg [1971]. Originally this algorithm appeared in Salkin [1969], Marsten [1971], [1972], Michaud [1972], Pierce [1

and Pierce and Lasky [1973]. Cutting plane techniques are described by Bellmore and Ratliff [1971], Bouse, Nelson, and Rado [1966], and Salkin and Koncal [1970], [1971]. A group of theoretic procedure is in Thiriez [1969], and heuristic methods are given by Ignizio and Harnett [1972] and Rubin [1973]. Also, a simplex type procedure, based on a result conjectured by Andrew, Hoffman and Krabek [1968], has been formulated by Balas and Padberg [1972], [1972]. Survey articles describing application algorithms and computations include Balinski [1965], Balinski and Spielberg [1969], Christofides and Korman [1973], Garfinkel and Nemhauser [1972], [1972], and Salin and Saha [1972].

Recent successful applications of the set partitioning models are given in Marsten and Shepardson [1981]. A computational study of the set covering algorithms using cutting planes are discussed in Balas and Ho [1980]. Recent work on heuristic algorithms for the set covering problems includes Baker [1981], Vasko and Wilson [1984], [1986].

5.4.1 Application of Set Covering Problem :

(a) Set Covering and Networks :

Certain problems one a network with undirected arcs (an arc which can be traversed in either direction) can be posed

as either a set covering or a set partitioning problem, where the constraint matrix has particular structure. Recall that a network is a collection of nodes and arcs joining them. An arc joining the nodes r and s will be denoted by (r,s) , where r and s are called the end points of the arc.

(b) The Node Covering Problem (Balinskin [1965], Balinski and Quandt [1964])

Define a cover to be subset of arcs in the network such that each node of the network is an end point of atleast one of the arcs in the subset. The simple covering problem is to find a cover with a minimum number of arcs.

To model the problem, number the arcs and let x_j represent the jth arc of the network, where $x_j = 1$ if the jth arc is in the cover and $x_j = 0$ otherwise. To make sure that each node i in the network is an end point of at least one arc in the cover, we must have $x_j = 1$ where the jth arc has node i as one of its end points. Thus, define $E = (e_{ij})$ by letting $e_{ij} = 1$ if the ith node in the network is an end point of the jth arc, and $e_{ij} = 0$ otherwise. The problem of finding a minimal cover is then a set of covering problem in which the costs are equal to 1.

Note that if there is a cost associated with every arc, we have a minimal cost covering problem which is also a set covering problem.

(c) The Matching Problem (Balinski [1965], Balinski and Quandt [1964])

A matching for a network is a subset of the arcs in the network such that no two arcs in the subset have a common end point. That is, a subset of arcs is a matching if and only if each node in the network has at most one arc in the subset incident to it. The simple matching problem is to find a matching which has a maximum number of arcs.

To model this problem, we define the binary variables (x_j) and the E matrix as in the minimal covering problem. As a maximum matching is sought, the objective function is to maximise $\sum_j x_j$ and since each node is to be the end point of at most one arc, the constraints are $Ex \leq e$. Thus the simple matching problem is to

$$\text{maximise} \quad \sum_j x_j$$

$$\text{subject to} \quad Ex \leq e$$

$$\text{and} \quad x_j = 0 \text{ or } 1 \quad (j=1,\dots),$$

which is a set partitioning problem with a maximization objective function. If there is weight c_j associated with each arc, the objective function becomes $\sum_j c_j x_j$ and we have a weighted matching problem.

(d) Disconnecting Paths (Bellmore, Greenberg, and Jarvis [1970])

Given a network, define a path from a specified node s to a given node t as the collection of nodes and arcs :

$$s, i_1, i_2, \ldots, i_r; t \quad \text{where} \quad s \neq i_1 \neq i_2 \neq \ldots \neq i_r \neq t$$

Suppose all the paths in a network are known, and also assume t there is a cost associated with removing an arc from the network. The problem is to find a set of arcs which, if removed from the network, will disconnect all the paths from s to t with the total cost of the removed arcs minimized.

To model the problem, number the arcs and paths in the network. Define the binary variable $x_j = 1$ if arc j is to be removed, and let $x_j = 0$ otherwise. The constraint matrix is found by letting $e_{ij} = 1$ if the jth arc is in the j^{th} path and $e_{ij} = 0$ otherwise. The situation is then represented by a set covering problem in which the $Ex \geq e$ inequalities ensure

that an arc is omitted from each path.

(e) Airline Crew Scheduling (Spitzer [1961]):

In this situation, we have to assign a set of m flights which must be flown over a given time period to a set of N crews so that the cost of operation is minimised. Subject to time, geography, crew ability, and other limitations, the combinations of flights that each crew may take are enumerated. Let $x_{k(t)}$ denote the tth way that the kth crew can make the flights, $x_{k(t)} = 1$ if the kth crew is assigned the series of flights designated in the tth combination, and $x_{k(t)} = 0$ otherwise. Let $e_{ik(t)} = 1$ if the ith " flight leg " can be assigned to kth crew in the tth combination, and let $e_{ik(t)} = 0$ otherwise. Since each flight leg must be assigned to exactly (or at least) one crew, we have the standard constraints

$$\sum_k \sum_t e_{ik(t)} x_{k(t)} = (\text{or } \geq)\ 1 \quad (i=1,\ldots,m) \qquad \ldots(5.25$$

where $x_{k(t)} = 0$ or 1 and $e_{ik(t)} = 0$ or 1. Let $c_{k(t)}$ be the cost of the tth combination of the kth crew. So the problem is to determine the value of the variables $x_{k(t)}$ such that $\sum_k \sum_t c_{k(t)} x_{k(t)}$ is minimised and the constraints (5.25) are satisfied.

Note that in this model we usually have the requirements that each crew be assigned to only one combination or series of flights. Hence the additional constraints

$$\sum_t x_{k(t)} \leq 1 \quad (k=1,2,\ldots, N) \qquad \ldots(5.26)$$

may appear in the model. Observe, however, that these inequalities are of the form $\sum_t x_{k(t)} + s_k = 1$ for each k, where s_k is a non-negative slack variable which must be or or 1 whenever all the $x_{k(t)}$'s are binary. Thus, if these constraints are added and there is an equality in (5.25), we still have a set partitioning problem.

(f) Truck Scheduling (Balinski and Quandt [1964]):

In this situation, there is a central warehouse with trucks, and m clients at different locations who have placed orders for goods at the warehouse. The problem is to transport the goods by trucks to the customers at a minimum cost of filling the orders. Given the m orders, the possible delivery schedules are determined on the basis of truck capabilities (e.g. size) and route constraints. For each truck, we may list all possible schedules and assign a weight (or cost) to each. Thus, let N denotes the total number of route possibilities

and let $c_{k(t)}$ denote the cost of the delivery associated with the tth possibility of kth truck. Suppose $x_{k(t)} = 1$ if the tth schedule for truck k is selected and let $x_{k(t)} = 0$ otherwise. Also, let $e_{ik(t)} = 1$ if the ith order is to be delivered in the tth schedule of truck k and let $e_{ik(t)} = 0$ otherwise, then the problem of transporting goods to m customers is a set partitioning problem where constraints of the form (5.26) ensure that each truck is used at most once.

(g) Political Redistricting (Vaszher [1968], Garfinkel and Nemhauser [1970]) :

Political districting is a process by which an area (e.g. state) is partitioned into smaller areas, each of which is assigned a single representative. Let m be the number of basic population units (counties , census tracts etc.). A district is a combination of these units which meets population, compactness, and other requirements. The various ways the population units can compose a district can be enumerated. Let n be the number of possible districts. The problem is to select k ($k \leq n$) districts so that every population unit belongs to exactly one district. We define x_j to be 1 if the jth district is in the redistricting

plan and o otherwise. Further, let $e_{ij} = 1$ if the ith population unit is in the jth districts and let $e_{ij} = o$ otherwise. Then if c_j is defined to be some measure of unacceptability of the jth district, the problem of finding the districts which will cover all the population units at a minimal measure of unacceptability is a set partitioning problem with the additional constraint $\sum_{j=1}^{n} x_j = k$.

(h) Information Retrieval (Day [1965]) :

The problem is to retrieve a given set of m requests for information from a set of n files so that the length of the search is minimized. Here the cost c_j corresponds to the length of the jth file, x_j is equal to 1 if the jth file is selected and x_j is a otherwise, also $e_{ij} = 1$ means that the ith information requested is in the jth file, and $e_{ij} = o$ if it is not.

5.4.2 Algorithms

There are two techniques for solving the set covering and set partitioning problem. Both are search algorithms and have performed reasonably well in computations. A search algorithm is a basic approach coupled with a point or node algorithm. A fairly detailed discussion on a search algorithm for the set

partitioning problem is given in Garfinkel and Nemhauser [1969] and Marsten and Shepardson [1981]. A search algorithm for solving the set covering problem is found in Lemke, Salkin and Spielberg [1971], and Salkin [1969]. Heuristics for set covering problems are presented in Baker [1981], Balas and Ho [1980], Vasko and Wilson [1984], [1986].

5.5 THE TRAVELLING SALESMAN PROBLEM

The travelling salesman problem is the most celebrated of all discrete optimization problems. The need to solve large traveling salesman problems arises in numerous applied contexts among which are problems in vehicle routing, computer writing, and job-shop scheduling. Early history of the problem is found in Flood [1956]. Dantzig, Fulkerson and Johnson [1954] presented a solution of a large scale problem. Gonzalez [1962] have solved the problem by dynamic programming. Held and Karp [1962] have solved problem up to 13 cities by dynamic programming. They developed an approximation that seems to work but does not guarantee an optimal tour. Dantzig, Fulkerson and Johnson [1959] have applied combinatorial approach to the problem. Dantzig, Fulkerson, and Johnson [1954], Miller, Tucker and Zemlin [1963] have formulated the traveling salesman problem as an integer

programming problem and cutting plane methods are applied to solve it in which further developments are made by Miliotis [1976], [1978]. A heuristic is given by Karg and Thompson [1964], and Lin and Kernighan [1973]. Subtour elimination algorithms are discussed in Bellmore and Malone [1971]. Christofides and Eilon [1972] have developed bounds for the large scale traveling salesman problem. Garfinkel [1973] partitioned the feasible set in a branch and bound algorithm for the asymmetric traveling salesman problem. Basaraa and Goode [1977] applied duality to the problem. An interesting and fairly detailed description of applications of the problem is given in Lenstra, J., and Rinnooy Kan A [1975]. Computational study are found in Padberg and Hong [1977], and Smith, Srinivasan, and Thompson [1977], Lagrangian approach to the problem is given by Balas and Christofides [1981]. A detailed discussion on the various algorithmic developments are found in Parker and Rardin [1985]. Many heuristic procedures have been developed for solving the traveling salesman problem. Rosenkrantz [1977] gave an analysis of several of them. Survey articles include Golden [1980], Lawler [1985] and Yang [1988].

Here we shall present mathematical formulation of the problem and then two classical solution techniques will be referred to.

5.5.1 Definition and Mathematical Formulation of the Problem :

The traveling salesman problem is easy to define : A salesman, starting in one city, wishes to visit each of the n-1 cities once and only once and return to the start. In what order should he visit the cities to minimise the total distance traveled ? For 'distance' we can substitute time, cost, or other measure of effectiveness as desired. Distance or costs between all city pairs are presumed known.

Before introducing the formulation, a definition of a tour and a subtour is given. A solution to the traveling salesman problem is said to constitute a tour if, starting from an arbitrary city, say o, every city is visited exactly once before returning to city 0. (The distance matrix d_{ij} is assumed to satisfy the triangular inequality.) A subtour may be defined as a tour comprising k cities, where k is strictly less than n. Thus a subtour is not a feasible solution to the problem.

Suppose we label the home city as city 0 and n+1. (Then we may think of the salesman's initial location as city 0 and the desired final location as city n+1). Also introduce the zero-one variable x_{ij} ($i=0,1,\ldots,n$, $j=1,\ldots,n+1, i \neq j$), where

$x_{ij} = 1$ if the salesman travels from city i to j, and $x_{ij} = 0$ otherwise. To guarantee that each city (except city 0) is entered exactly once, we have

$$\sum_{i=0}^{n} x_{ij} = 1 \qquad (j=1,\ldots,n,\ i \neq j)$$

Similarly, to ensure that each city (except city n+1) is left exactly once, we have

$$\sum_{j=1}^{n+1} x_{ij} = 1 \qquad (i=0,1,\ldots,n,\ i \neq j)$$

These constraints however do not eliminate the possiblity of subtours of " loops ". One way of eliminating the subtour possiblity is to add the constraints

$$\alpha_i - \alpha_j + (n-1)\, x_{ij} \leq n \qquad (i=0,1,\ldots,n, j=1,\ldots,n+1, i \neq j)$$

where α_i is a real number associated with city i.

To show that a solution containing loops cannot satisfy these constraints, consider any subtour except the one containing the home city. Then, if we sum the inequalities corresponding to the $x_{ij} = 1$ around the loop, the $\alpha_i - \alpha_j$ cancel each other and we are left with $(n+1)N \leq nN$, where N is the number of arcs in a subtour, which is a contradiction.

On the other hand, to see that these constraints may be satisfied when there are no subtours, define $\alpha_0 = 0, \alpha_{n+1} = n+1$, and let $\alpha_i = k$ if city i is the kth city visited on the tour. Then, when $x_{ij} = 1$, we have $\alpha_i - \alpha_j + (n+1) = k-(k+1)+(n+1) = n$, since $1 \leq \alpha_j \leq n+1$ $(i=1,\ldots,n+1)$, the difference $\alpha_i - \alpha_j$ is always $\leq n$ (all i,j), and thus the constraints are satisfied when $x_{ij} = 0$.

To complete the model we wish to minimize the total distance $\sum_{i=0}^{n} \sum_{j=1}^{n+1} d_{ij} x_{ij}$. Or an integer programming formulation of the traveling salesman problem is to find variable x_{ij} and arbitrary real numbers α_i which

$$\text{minimize} \quad \sum_{i=0}^{n} \sum_{\substack{j=1 \\ j\neq 1}}^{n+1} d_{ij} x_{ij}$$

$$\text{subject to} \quad \sum_{i=0}^{n} x_{ij} = 1 \quad (j=1,\ldots,n+1, i\neq j) \qquad \ldots(5.\text{-}\,)$$

$$\sum_{j=1}^{n+1} x_{ij} = 1 \quad (i=0,1,\ldots,n, i\neq j) \qquad \ldots(5.\text{-}\,)$$

$$\alpha_i - \alpha_j + (n+1)x_{ij} \leq n \quad (i=0,1,\ldots,n, j=1,\ldots,n+1, i\neq j)$$

$$\text{and} \quad x_{ij} = 0 \text{ or } 1 \quad (i=0,1,\ldots,n, j=1,\ldots,n+1, i\neq j)$$

where $x_{0,n+1} = 0$ (since $x_{ij} = 0$ for $i = j$). This formulation originally appeared in Tucker [1960]. In the absence of the constraint (5.27) the problem (5.28) is a case of general assignment problem. Unfortunately, there is no guarantee that the optimal solution of the assignment model will be a tour. Most likely such a solution will consists of subtours. Two cutting plane methods for the solution of the problem may be seen in Dantzig, Fulkerson and Johnson [1954], and Miller, Tucker and Zemlin [1960]. The branch abd bound method may be seen in Parker and Rardin [1983]. These methods are aimed to showing how the tour restriction is imposed explicitly so that the optimal solution is a tour.

REFERENCES

1- Agin, N., 'Optimum Seeking with Branch and Bound', Management Science 13(4), 176-185(1966).

2- Andrew, G., T.Hoffman, and C.Krabek, 'On the Generalized Set Covering Problem,' Control Data Cooperation, Data Centres Division, Minneapolis.(Paper presented at the ORSA meeting in San Francisco, 1968).

3- Armour, G., and E. Buffa, 'A Heuristic Algorithm and Simulation Approach to Relative Location of Facilities,' Management Science 9(2), 294-309(1965).

4- Arnold, L., and M.Bellmore, 'Iteration Skipping in Primal Integer Programming', Operations Research 22(1), 129-136(1974)

5- Austin, L.M., and Ghandforoush,P., 'An advanced Dual Algorithm with Constraint Relaxation for All-Integer Programming', Naval Research Logistic Quarterly 30(1), 133-144 (1983).

6- Baker, E., 'Heuristic Algorithms for the weighted Set Covering Problems', Computers and Operations Research 8, 303-310 (1981).

7- Balas, E., 'Linear Programming with Zero-One Variables' (In Rumanian), Proceedings of the Third Scientific Session on Statistics, Bucharest, December 5-7,1963.

8- Balas, E., 'An Additive Algorithm For Solving Linear Programs with Zero-one Variables', Operations Research 13(4),517-548 (1965).

9- Balas, E., 'Discrete Programming by the Filter Method', Operations Research 15(5), 915-1957 (1967).

10- Balas, E., 'A Note on the Branch and Bound Principle', Operations Research 16(2), 442-445 (1968).

11- Balas, E., 'The Intersection Cut-A New Cutting Plane for Integer Programming', Management Sciences Research Report No. 187, Carnegie-Mellon University, October 1969.

12- Balas, E., 'Intersection Cuts- A New Type of Cutting Plane for Integer Programming', Operations Research 19(1), 19-39 (1971).

13- Balas, E., and M.Padberg, 'On the Set Covering Problem', Operations Research 20(6), 1152-1161 (1972).

14- Balas, E., M.Padberg, 'On the set Covering Problem-II, An Algorithm', Management Science Research Report No.295, Carnegie-Mellon University, May 1972.

15- Balas, E., and A.C.Hoj,' Set Covering Algorithms Using Cutting Plane, Heuristics, and Subgradient Optimization A Computations Study', Mathematical Programming, Study 12,North-Holland, Amsterdam, pp 37-60(1980).

16- Balas, E.,and J.B.Mazzola,' Linearising Nonlinear 0-1 Programs', MSRR No.467, Carnegie-Mellon University,Pittsberg, PA, October 1980.

17- Balas, E., and J.B.Mazzola,' Non-linear 0-1 Programming : I.Linearisation Techniques', Mathematical Programming 30, 1-21 (1984).

18- Balinski, M.L., 'Fixed Cost Transportation Problems', Naval Research Logistic Quarterly 11,41-54 (1961).

19- Balinski, M.L.,' On Finding Integer Solutions to Linear Programs', Mathematica, Princeton, New Jersey,May 1964.

20- Balinski, M.L.,'Fixed Cost Transportation Problems', Naval Research Logistic Quarterly 11,41-54 (1961).

21- Balinski, M., and R.Quandt,' On an Integer Program for a Delivery Problem', Operations Research 12(2), 300-304 (1964).

22- Balinski, M.,and K.Spielberg,' Methods for Integer Programming: Algebraic, Combinatorial and Enumerative', in Progress in Operations Research Vol.III (ed.Aronofsky), New York,John Wiley and Sons, 1969.

23- Balinski, M.,and P.Wolfe,' On Benders Decomposition and a Plant Location Problem', Mathematica Working Paper ARO-27, 1963.

24- Baumol, W., and P.Wolfe,' A Warehouse Location Problem', Operations Research 6(2), 252-263(1968).

25- Bazara, M.S.,and J.J.Goode,'A Cutting Plane Algorithm for the Quadratic Set Covering Problem', Operations Research 23, 150-158 (1977).

26- Beale, E.,' A Method of Solving Linear Programming Problems When Some But Not all of the Variables Must Take Integral Values', Statistical Techniques Research Group, Technical Report No.19, Princeton University, July 1958.

27- Beale, E., 'Survey of Integer Programming', Operations Research Quarterly 16, 219-228 (1965).

28- Bellman, R., 'Some Applications of the Theory of Dynamic Programming - A Review', Operations Research 2(2),275-288 (1954).

29- Bellman, R., 'Notes on the Theory of Dynamic Programming IV-Maximization over Discrete Sets,' Naval Research Logistic Quarterly 3,67-70 (1956).

30- Bellman,R., 'Comment on Dantzig's Paper on Discrete Variable Extremum Problems', Operations Research 5(5), 723-724(1957).

31- Bellman,R.,' Dynamic Programming' Princeton University Press 1957.

32- Bellman, R., and S. Dreyfus, 'Applied Dynamic Programming', Princeton University Press, 1962.

33- Bellmore, M., H.Greenberg, and J.Jarvis,' Multi-Commodity Disconnecting Sets', Management Science 16(6), 427-433(1970).

34- Bellmore, M., and J.C.Malone,' Pathology of Traveling Salesman Subtour Elimination Algorithms', Operations Research 19, 278-307 (1971).

35- Bellmore, M.,and H.Ratliff, 'Optimal Definite of Multi-Commodity Networks', Management Science 18(4),174-185(1971).

36- Benders, J., 'Partitioning Procedures for Solving Mixed Variables Programming Problems', Numerische Mathematik 4, 238-252 (1962).

37- Ben-Israel, A.,and A.Charnes,' On Some Problem of Diophantine Programming', Cahiers du Centre d' Etudes de Recherche Operationnelle (Bruxelles) 4, 215-280(1962).

38- Bertier, P., and B. Roy,' A Solution Procedure for a Class of Problems having combinatorial Structures', Operations Research Centre, University of California at Berkeley, September 1967.

39. Bowman, V., and F. Glover,' A Note on Zero-One Integer and Concave Programming', Operations Research 20(1),182-183(1972

40- Bradley, G.H.,P.L.Hammer, and L.Wolsey,'Coefficient Reduction for Inequalities in 0-1 Variables Research Report CORR 73-6. Department of Combinatorics and Optimization,University of Waterloo, Waterloo, Iowa.

41- Brown, W., 'A Generalised Plant Location Problem', Ph.D. thesis, Department of Operations Research, Case Western Reserve University, January 1973.

42- Burdet, C., 'A Class of Cuts and Related Algorithms in Integer Programming', Management Sciences Research Report No. 220, Carnagie-Mellon University, Pittsburgh, 1970.

43- Cabot, V.,' An Enumeration Algorithm for Knapsack Problems', Operations Research 18(2), 306-311 (1970).

44- Cabot, V., and A. Hunter, Jr.,' The Application of an Approach to Zero-one Programming to Knapsack Problems', Presented at the Joint ORSA/TIMS Meeting in San Francisco (1968).

45- Carstensent, P.J.,K.G.Murty,C.Perin,' Intermediate Feasibility in 0-1 Integer Linear Systems', Mathematical Programming 24, 219 - 277 (1985).

46- Charnes, A., D.Granot, F.Granot,' On Inter-section cuts in Interval Integer Linear Programming', Operations Research 25(2), 352-355 (1977).

47- Christofides, N.,and S. Eilon,'An Algorithm for the Vehicle Dispatching Problem', Operational Research Quarterly 20, 309- 318 (1969).

48- Christofides, N., and S. Korman,' A Computational Survey of Methods for the Set Covering Problem', Department of Management Science Report 73/2, Imperial College of Science and Technology (London),April 1973.

49- Ciochetto, F.,H.Swanson, J.Lee,and R.Woolsey,' The Lockbox Problem and Some St arting But True Computational Results for Large Scale Systems'. Paper presented at the 41st national ORSA meeting in New Orleans, April 1972.

50- Comlin, J., and K.Spielberg,' Branch and Bound Schemes for Mixed-Integer Programming,' IBM New York Scientific Centre Report No. 320-2972, May 1969.

51- Cooper, L.,' Heuristics Methods for Location-Allocation Problems', SIAM Review 6(1), 37-52 (1964).

52- Cooper, L.,and C. Drebes,' An Approximate Solution Method for the Fixed Charge Problem,' Naval Research Logistic Quarterly 14, 101-113 (1967).

53- Cooper, W. Murry,' A Survey of Methods for Pure- Nonlinear Integer Programming', Management Science 27(3),353-361 (1981).

54- Dakin, R., 'A Tree Search Algorithm for Mixed Integer Programming Problems', Computer Journal 8, 250-255 (1965).

55- Dantzig, G.B., 'Discrete Variable Extremum Problems, Operations Research 5, 266-277 (1957).

56- Dantzig, G.B.,' Notes on Solving Linear Programs in Integers', Naval Research Logistic Quarterly 6, 75-76 (1959).

57- Dantzig, G.B., 'Linear Programming and Extensions', Princeton University Press, Princeton, New Jersey.

58- Dantzig, G.B.,D.R.Fulkerson, S.M.Johnson,' Solution of a Large Scale Traveling Salesman Problem', Operations Research 2, 393-410 (1954).

59- Dantzig,G.B.,D.R.Fulkerson,S.M.Johnson,' On a Linear Programming Combinatorail Approach to the Traveling Salesman Problem', Operations Research 7, 58-66 (1959).

60- Davis, P.S., and T.L.Ray,' A Branch and Bound Algorithm for the Capacitated Facilities Location Problem', Naval Research Logistic Quarterly 16, 331-344 (1969).

61- Davis, R.E.,D.A.Kendrick,and M.,Weitzman,' A Branch and Bound Algorithm for 0-1 Mixed Integer Programming Problems', Operations Research 19(4), 1036-1044 (1971).

62- Denzler, D.R.,' An Approximative Algorithm for the Fixed Charge Problem', Naval Research Logistic Quarterly 16, 411-416 (1969).

63- Driebeek, N.J.,' An Algorithm for the Solution of Mixed Integer Programming Problems', Management Science 12, 576-587 (1966).

64- Dreyfus, S.,and K.Parther,' Improved Algorithms for Knapsack Problems', Operations Research Centre Report ORC 70-30, the University of California at Berkeley,1970.

65- Drysdale, J.K. and P.J.Sandiford,' Heuristic Warehouse Location- A Case History Using a New Method', Can.Operational Res. Soc. J. 7,45-61 (1969).

66- Dudzinski, Krysztof, Walukiewicz, Stanislaw,' Exact Methods for the Knapsack Problem and Its Generalisations', European Journal of Operations Research 28(1),3-21 (1987).

67- Dwyer, P.,' The Direct Solution of the Transportation Problem with Reduced Matrices,' Management Science 13(1), 77-98 (1966).

68- Dyer, M.E., 'An O(n) Algorithm for the Multi-Choice Knapsack Linear Program', Mathematical Programming 29, 57-63 (1984).

69- Efroymson, M.A.,and T.L.Ray,' A Branch and Bound Algorithm for Plant Location', Operations Research 14, 361-368 (1966).

70- Eiselt, H.A.,' Continuous Maximum Knapsack Problems with GLB Constraints', Mathematical Programming 36(1),114-121 (1986).

71- Everett, H., 'Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources', Operations Research 11,399-417 (1963).

72- Faaland, B.,' Solution of Value-Independent Knapsack Problem by Partitioning', Operations Research 21(1), 332-333(1973).

73- Feldman, E.,F.A.,Lehrer,and T.L.Roy,' Warehouse Location Under Continuous Economies of Scale', Management Science 12 (9), 670-684 (1966).

74- Fisk, J., and P.McKeown,' The Pure Fixed Charge Transportation Problem', Naval Research Logistic Quarterly 26, 631-641 (1979).

75- Fleischmann, B., ' Computational Experience With the Algorithm of Balas', Operations Research 15, 153-155(1967).

76- Flood, M., 'The Trafeling Salesman Problem', Operations Research 4(1), 61-75 (1956).

77- Freeman, R. J.,' Computational Experience With a 'Balasian' Integer Programming Algorithm', Operations Research 19, 1747 - 1751 (1973).

78- Garfinkel,R.S.,'On Partitioning the Feasible Set in Branch and Bound Algorithm for the Asynptotic Traveling Salesman Problem', Operations Research 21,340- 342 (1973).

79- Garfinkel, R.S.,and G.L.Nemhauser,' The Set Partitioning Problem:Set Covering with Equality Constraints', Operations Research 17, 848- 856 (1969).

80- Garfinkel, R.S.,and G.L.Nemhauser,' Optimal Political Districting by Implicit Enumeration Techniques',Management Science 16, B 495-B 508 (1970).

81- Garfinkel,R.S.,and G.L.Nemhauser,' Optimal Set Covering : A Survey. In Perpectives on Optimization: A Collection of Expository Articles (A.M. Geoffrion,ed.)',pp 164-183 Addison-Wesley, Reading, Massachusetts, 1972.

82- Garfinkel,R.S.and G.L.Nemhauser,' Integer Programming', Wiley, New York 1972.

83- Geoffrion, A.,' Integer Programming by Implicit Enumeration and Balas' Method', SIAM Review 9(2), 178-190 (1967).

84- Geoffrion,A.,' An Improved Implicit Enumeration Approach for Integer Programming', Operations Research 17(3), 437-454 (1969).

85- Geoffrion, A.,' The Capacitated Facility Location Problem with Additional Constraints', Operations Research Working Centre discussion paper, the University of California as Los Angeles, February 1973.

86- Geoffrion, A.,and R. Marsten,' Integer Programming : A Framework and State of the -Art Survey', Management Science 18(9), 465-491 (1972).

87- Ghandforoush, P. and L.M.Austin,' A Primal-Dual Cutting-Plane Algorithm for All-Integer Programming', Naval Research Logistic Quarterly, 28(4), 559 - 566 (1981).

88- Gilmore, P.C., and R.E.Gomory,' A Linear Programming Approach to the Cutting Stock Problem', Operations Research 9, 849-859 (1961).

89- Gilmore,P.C.,and R.E.Gomory,' A Linear Programming Approach to the Cutting Stock Problem-Part II', Operations Research 11, 863-888 (1963).

90- Gilmore,P.C. and R.E.Gomory,' Many Stage Cutting Stock Problems of Two or More Dimensions', Operations Research 13, 94 - 120 (1965).

91- Gilmore,P.C., and R.E.Gomory,' The Theory of Computation of Knapsack Functions', Operations Research 14,1045-1074 (1966).

92- Glover,F.,' A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem', Operations Research 13(6), 879-919 (1965).

93- Glover, F.,' A Note on Linear Programming and Integer Feasibility', Operations Research 16,1212-1216 (1968).

94- Glover, F.,' Improved Linear Representations of Discrete Mathematical Programs', Management Science Report No.72-8, the University of Colorado at Boulder, February 1972.

95- Glover, F.,' Convexity Cut and Cut Search', Operatio ns Research 21, 123- 134 (1973).

96- Glover, F., and D. Klingman,' Mathematical Programming Models and Methods for the Journal Selection Problem ', Operations Research 21(1), 141-155 (1973).

97- Glover, F., and D. Sommer,' Pitfalls of Rounding in Discrete Management Decision Problems', Management Science Report No. 72-2 University of Colorado, Boulder, 1972.

98- Glover, F., and R.Woolsey,' Further Reduction of Zero-one Polynomial Programming Problems to Zero-one Linear Programming Problems', Operations Research 21(1), 156-161 (1973).

99- Glover,F.,and R.Woolsey,' Converting the 0-1 Polynomial Programming Problem to 0-1 Linear Program', Operations Research, 21, 156-161 (1973).

100- Glover,F., and S.Zionts,' A Note on the Additive Algorithm of Balas', Operations Research 13(4), 546-549 (1965).

101- Golomb, S.,and L.Baumert,' Backtrack Programming', Journal of the Association for Computing Machinery 12(4), 516-524 (1965).

102- Golden,B.,L.Bodin,T.Doyle,and W.Jr.Stewart, 'Approximate Traveling Salesman Problem Algorithms', Operations Research 28, 694 - 711 (1980).

103- Gomory, R.,' Outline of an Algorithm for Integer-Solutions to Linear Programs', Bull.Amer.Math.Soc.64, 275-278 (1958).

104- 'An Algorithm for the Mixed Integer Problem', RM-2597 Rand Corporation, July 1960.

105- Gomory, R., and A. Hoffman, 'On the Convergence of an Integer Programming Process', Naval Research Logistic Quarterly 10(1), 121-125 (1963).

106- Gonzales,R.,' Solution to the Traveling Salesman Problem by Dynamic Programming on the Hypercube', Operations Research Centre Technical Report No. 18, Massachusetts Institute of Technology, 1962.

107- Granat, F.,and P. Hammer,' On the Role of Generalized Covering Problems', Centre of Cybernetic Study Research Report CS 96, the University of Texas at Austin,Sept.1972.

108- Gray, P., 'Mixed Integer Programming Algorithms for Site Selection and Other Fixed Charge Problems Having Capacity Constraints', Ph.D. thesis, Stanford University, Nov.1967.

109- Gray, P., 'Exact Solution of the Fixed Charge Transportation Problem', Operations Research 19(6), 1529-1538 (1971).

110(a)- Greenberg, H.,' An Algorithm for the Computation of Knapsack Functions', Journal of Mathematical Analysis and Applications 26, 159-162 (1969).

110(b)- Guignard, M., and K.Spielberg,' Search Techniques with Adaptive Features for Certain Integer and Mixed Integer Programming Problems', Proceedings of the Fifth IFIP Congress, 141-146, 1968.

110(c)- Guignard,M., and Spielberg,' The State Enumeration Method for Mixed Zero-One Programming', IBM Philadelphia Scientific Centre Report No. 320-3000, 1971.

110(d)- Guignard,M.,and K.Spielberg', A Realization of the State Enumeration Procedure', IBM Philadelphia Scientific Centre Working Paper, 1971.

111- Guha, D.,' The Set Covering Problem with Equality Constraint Operations Research 21(1), 348-351 (1973).

112- Gupta, K. Omprakash, and A. Ravindran,' Branch and Bound Experiments in Convex Integer Programming', Management Science 31(12), 1533-1546 (1985).

113- Gupta, and A. Ravindran, 'Nonlinear Integer Programming Algorithms : A Survey', Opsearch 20, 189-206 (1983).

114- Guignard, M., and K.Spielberg,' The State Enumeration Method for Mixed Zero-One Programming', IBM Philadelphia Scientific Centre Report No. 320-3000 February 1971.

115- Gulley, D.,H.Swanson,and R. Woolsey,' On Not Searching for the Multipliers in Knapsack Problems', Colorado School of Mines, Golden, Colorado.

116- Halfin, S.,' Arbitrarily Complex Corner Polyhedra are Dense in R^n, Bell Telephone Laboratories Paper, Holmdel, New Jersey, 1972.

117- Hammer,P.,and S.Rudeanu, Boolean Methods in Operations Research and Related Areas, New York, Springer-Verlag,1968.

118- Hansman, B.,' Operations Research in the National Planning of Underdeveloped Countries', Operations Research 9(1), 203-248 (1961).

119- Held, M., and R.M.Karp, (1962), See Parker and Rardin (1983).

120- Hirsch, W. and G.Dantzig,' The Fixed Charge Problem' Naval Research Logistic Quarterly 15(3), 413-424 (1968), Originally appeared in 1954.

121- House, R.,L.Nelson, and T. Rado,' Computer Studies of a Certain Class of Linear Integer Programs',; in Recent Advances in Optimisation Techniques (eds. Lavi and Vogl), New York, John Wiley and Sons, 1966.

122- Ibaraki, T.,T.Hasegawa,K.Teranaka,and J.Iwase,' The Multiple Choice Knapsack Problem', Journal of the Operations Research Society of Japan 21, 59-95 (1978).

123- Ignizio, J., and R. Harnett,' Heuristically Aided Set Covering Algorithms', University of Alabama, Huntsville, December 1972.

124- Jambekar, A.,and D.Steinberg,' Computational Experience with a New Algorithm for 0-1 Integer Programming', School of Business and Engineering Administratio n Paper, Michigan Technological University, Houghton, Michigan, May 1973.

125- Jain, A., Private Communication (at Indian Explosives Ltd. Calcutta, India), June 1972.

126- Jeroslow, R.,' On the Unlimited Number of Faces in Integer Hulls of Linear Problems with Two Constraints', Department of Operations Research Technical Report No. 67, Cornell University, April 1969.

127- Jeroslow, R.,' Comments on Integer Hulls of Two Linear Constraints', Operations Research 19(4), 1061-1069 (1971).

128- Jeroslow, R., and K. Kortanek, 'Dense Sets of Two Variable Integer Programs Requiring Arbitrarily Many Cuts by Fraction Algorithms,' Management Sciences Research Report 174, Carnegi Mellon University, 1969.

129- Jeroslow, R.,and K.Kortanek,' On an Algorithm of Gomory', SIAM Journal of Applied Mathematics 21(1), 55-60 (1971).

130- Johnson, E., and K.Spielberg,' Inequalities in Branch and Bound Programming', IBM Thomas J.Watson Research Centre Report RC 3649, Yorktown Heights, New York, December 1971.

131- Jones, A.P., and R.M.Soland,' A Branch and Bound Algorithm for Multilevel Fixed Charge Problems', Management Science 16(1), 67-76 (1969).

132- Karg, R.L.,and G.L. Thompson,' A Heuristic Approach to Solving Traveling Salesman Problem', Management Science 10, 1130-1136 (1964).

133- Kennington, J.E.,and D. Fyffe,' A Note on Quadratic Programming Approach to the Solution of 0-1 Linear Integer Programming Problem', School of Industrial and System Engineering Report, Georgia Institute of Technology,1972.

134- Kennington, J.L.and V.E.Unger,'The Group Theoretic Structure in the Fixed Charge Transportation Problem', Operations Research 21(5), 1142-1153 (1973).

135- Kennington, J.L.,and V.E.Unger,' A New Branch and Bound Algorithm for the Fixed Charge Transportation Problem ', Management Science 22, 1116-1126 (1976).

136- Khumawala, B.M ' An Efficient Branch and Bound Algorithm for the Warehouse Location Problem', Management Science 18 (12), 718 - 731 (1972).

137- Kolesar, P.J.,' A Branch and Bound Algorithm for the Knapsack Problem', Management Science 13, 723-735 (1967).

138- Krafft, D.,and T. Hill,' A Lagrarian Formulation of the Journal Selection Model', School of Industrial Engineering Technical Report, Purdue University, 1971.

139- Kraft, D., and T. Hill,' The Journal Selection Problem in a University Library System', Management Science 19(6), 613-626 (1973).

140- Kuehn, A.,and M.Hamburger,'A Heuristic Program for Locating Warehouses', Management Science 10(4), 643-666 (1963).

141- Kuhn, H., and W. Baumol,' An Approx imate Algorithm for the Fixed Charge Transportation Problem', Naval Research Logistics Quarterly 9(1), 1-16 (1962).

142- Land, A., and A. Doig,' An Automatic Method of Solving Discrete Programming Problem', Econometrica 28(3) , 497-520 (1960).

143- Lasdon, L., Optimisation Theory for Large Systems, New York, McMillan , 1970.

144- Lawler, E.,and M.Bell,' A Method for Solving Discrete Optimisation Problems', Operations Research 14(6), 1098-1112 (1966).

145- Lawler, E.,and D. Wood,' Branch and Bound Methods : A Survey', Operations Research 14(4), 699-719 (1966).

146- Lawler, E.,J.K.Lenstra, A.H.G.Rinnooy Kan, and D.B. Shmoys', The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimisation' (Wiley,1985),214-239.

147- Lemke, C.,and K. Spielberg,' Direct Search Algorithms for Zero-One and Mixed Integer Programming', Operations Research 15(5), 892-914 (1967).

148- Lemke,C.,H.Salkin,and K.Spielberg,' Set Covering by Single Branch Enumeration with Linear Programming Subproblems', Operations Research 19(4), 998-1022 (1971).

149- Lenstra, J.,and A.H.Rinnooy Kan (1975), See Parker and Rardin (1983).

150- Lu, S.M. and A.C.William,' Roof Duality for Polynomial 0-1 Optimisation', Mathematical Programming 37,357-360(1987)

151- Manne, A.S.,' Plant Location Under Economic of Scale-Decentralisation and Computation', Management Science 11, 213-235 (1964).

152- Markowitz, H.M.,and A.S.Manne,' On the Solution of Discrete Programming Problems, Econometrica 25,84-110 (1957).

153- Marsten, R.E.,' An Implicit Enumeration Algorithm for the Set Partitioning Problem with Side Constraints', Ph.D. thesis, University of California, Los Angles ,1971.

154- Marsten, R.E.,'An Algorithm for Large Set Partitioning Problems', Centre for Mathematical Studies in Economics and Management Science,Discussion Paper No.8,Northwestern University 1972.

155- Marsten, R.E. and Shepardson,F.,' Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model : Recent Successful Applications', Networks 11,165-177(1981).

156- Mathews, G.,' On the Partition of Numbers', Proceedings of the London Mathematical Society 28, 486-490 (1897).

157- Mathis, J.,' A Counter Example to the Rudimentary Primal Integer Programming Algorithms', Operations Research 19(6), 1518-1522 (1971).

158- McKeown, P.G.,' A Branch and Bound Algorithm for Solving Fixed Charge Problem', Naval Research Logistic Quarterly 28, 607-617 (1981).

159- Michaud, P.,' Exact Implicit Enumeration Method for Solving the Set Partitioning Problem', IBM Journal of Research and Development 16(6), 573-578 (1972).

160- Miliotis (1976),(1978), See Parker and Rardin (1983).

161- Miller, C.,A.Tucker, and R.Zemlin,' Integer Programming Formulations of Traveling Salesman Problems'. J.Assoc. Comput.Mach. 7, 326-329 (1960).

162- Mitten, L.,' Branch and Bound Methods:General Formulation and Properties', Operations Research 18(1),24-34 (1970).

163- Padberg,M.,and Hang (1977), See Parker and Rardin (1983).

164- Parker, R.G.,and R.L.Rardin,' The Traveling Salesman Problem: An Update of Research', Naval Research Logistic Quarterly, 30(1), 69-96 (1983).

165- Peterson, C.C.,' Computational Experience with Variants of Balas Algorithm Applied to the Selection of R and D Projects', Management Science 13, 736-750 (1967).

166- Peterson, C.C.,'A Note on Transforming the Product of Variables to Linear Form in Linear Programs', Working Paper, Purdue University, (1971).

167- Pierce, J.,' Application of Combinatorial Programming to a Class of all Zero-One Integer Programming Problems', Management Science 15(1), 191-209 (1968).

168- Pierce, J.,and J.Lasky,' Improved Combinatorial Programming Algorithms for a Class of All Zero-One Integer Programming Problems', Management Science 19(5), 528-543 (1973).

169- Pinkus, C.,D.Gross,and R.Soland,' Optimal Design of Multiactivity Multifacility Systems by Branch and Bound', Operations Research 21(1), 270-285 (1973).

170- Raghavachari, M.,' On Connections Between Zero-One Integer Programming and Concave Programming Under Linear Constraints' Operations Research 17(4), 680-684 (1969).

171- Raghavachari, M.,' Supplement(, Operations Research 18(3), 564-565 (1970).

172- Rardin,R. and V.Unger,' A Surrogate Constraint for 0-1 Mixed Integer Programs', Paper presented at 43rd National ORSA meeting, 1975.

173- Rasenkrantz, D.J.,R.E.Stearn, and P.M.Lewis,' An Analysis of Several Heuristic for the Traveling Salesman Problem', Siam J.Comput. 6, 563-581 (1977).

174- Ross, G.T.,and R. Soland,' A Branch and Bound Algorithm for the Generalized Assignment Problem', Mathematical Programming 8, 91- 103 (1975).

175- Rubin, D.,' The Neighbouring Vertex Cut and Cuts Derived with Gomory's Asymptotic Algorithm', Ph.D. Thesis, University of Chicago, 1970.

176- Rubin, J.,' A Technique for the Solution of Massive Set Covering Problems, with Application to Airline Crew Scheduling', Transportation Science 7(1), 34-48 (1973).

177- Rutten, D.,' Structured Integer Programs', paper presented at the 43rd National ORSA meeting, 1973.

178- Sa, G.,' Branch and Bound and Approximate Solutions to the Capacitated Plant Location Problem', Operations Research 17, 1005-1016 (1969).

179- Salkin, H.,' Enumerative Algorithms for Integer and Mixed Integer Programs', Department of Operations Research, Technical Memorandum No.162, Case Western Reserve University, 1969.

180- Salkin, H.,' On the Merit of the Generalized Origin and and Restarts in Implicit Enumeration', Operations Research 18(3), 549-555 (1970).

181- Salkin, H.,' A Note on Gomory's Fractional Cut', Operations Research 19, 1538-1541 (1971).

182- Salkin, H.,' A Brief Survey of Algorithms and Recent Results in Integer Programming', Opsearch 10(2), 81-123 (1973).

183- Salkin, H.,' Integer Programming', Addison-Wesley, 1975.

184- Salkin, H., and C. Dekluyver,' The Knapsack Problem : A Survey', Department of Operations Research Technical Report No. 281, Case Western Reserve University, 1972.

185- Salkin, H., and R. Koncal,' A Pseudo Dual All Integer Algorithm for the Set Covering Problem', Department of Operations Research Technical Memorandum No. 204, Case Western Reserve University, 1970.

186- Salkin,H.,and R. Koncal,' A Dual All Integer Algorithm (in Revised Simplex Form) for the Set Covering Problem', Department of Operations Research Technical Memorandum No. 250, Case Western Reserve University, 1971.

187- Salkin, H.,and J. Saha,' Set Covering :Uses, Algorithms, Results' Department of Operations Research Technical Memorandum No. 272, Case Western Reserve University,1972.

188- Salkin,H., and K. Spielberg,' Adaptive Binary Programming', IBM Pheladelphia Scientific Centre Report No. 320-2951,1968.

189- Shapiro, G., Shortest Route Methods for the Finite State Space Deterministic Dynamic Programming Problems', SIAM Journal of Applied Mathematics 16(6), 1232-1250 (1968).

190- Shapiro,G.,'Generalised Lagrange Multipliers in Integer Programming', Operations Research 19(1),68-77 (1971).

191- Shareshian,R.,' A Modification of the Mixed Integer Algorithm of N. Driebeek IBM Philadelphia Scientific Centre Report No. 2,1966.

192- Sinha, P.,and A.Zoltners,'The Multiple Choice Knapsack Problem', Operations Research 27,503-515 (1979).

193- Smith, Srinivasan, and Thompson (1977), See Parker and Rardin (1983).

194- Spielberg,K.,' On the Fixed Charge Transportation Problem', Proceedings of the 19th National ACM Conference ,1964.

195- Spielberg,K.,' Enumerative Methods for Integer and Mixed Integer Programming', IBM Philadelphia Scientific Centre Report No. 320-2958, 1968.

196- Spielberg,K.,' Algorithms for the Simple Plant Location Problem with Some Side Constraints', Operations Research, 17(1), 85-111 (1969).

197- Spielberg,K.,' Plant Location with Generalised Search Origin', Management Science 16(3), 165-178 (1969).

198- Spielberg,K.,' Minimal Preferred Variable Reduction for Zero-One Programming', IBM Philadelphia Scientific Centre Report No. 3 ,320-3013, 1972.

199- Speilberg, K.,' A Minimal-Inequality Branch-Bound Method', IBM Philadelphia Scientific Centre Working Paper, 1972.

200- Spitzer, M.,'Solution to the Crew Scheduling Problem', presented at the first AGIFORS Symposium,1961.

201- Suhl, U.H.,' Solving Large Scale Mixed Integer Programs with Fixed Charge Variables', Mathematical Programming 32, 165-182 (1985).

202- Taha, H.,' A Balasian-Based Algorithm for 0-1 Polynomial Programming', Research Report No. 70-2, the University of Arkansas, May 1970.

203- Taha, H., 'Integer Programming Theory, Applications, and Computations', Academic Press, INC. 1975.

204- Thiret, H.,' Air Line Crew Scheduling: A Group Theoretic Approach', Ph.D. Thesis, Massachusetts Institute of Technolog 1969.

205- Tomlin, J.,' Branch and Bound Methods for Integer and Non-Convex Programming', in Integer and Nonlinear Programming (ed.; Abadie), North-Holland, 1970.

206- Tomlin, J.,' An Improved Branch-and-Bound Method for Integer Programming', Operations Research 19(4),1070-1075(197

207- Tompkins, C.,' Group Theoretic Structures in the Fixed Charge Transportation Problem', Ph.D. Thesis, Georgia Institute of Technology, 1971.

208- Tuan, N.,' A Flexible Tree-Search Method for Integer Programming Problems', Operations Research 19(1),115-119(1971

209- Tucker, A.,' On Directed Graphs and Integer Programs', IBM Mathematical Research Project Technical Report, Princeton University, 1960.

210- Vasko, F.J.,and G.R.Wilson,' An Efficient Heuristic for Large Set Covering Problems', Naval Research Logistic Quarterly, 31,163-171 (1984).

211- Vasko, F.J.,and G.R.Wilson,' Hybrid Heuristics for Minimum Cardinality Set Covering Problems', Naval Research Logistic Quarterly, 33, 241-249 (1986).

212- Wagner, W.H.,' An Application of Integer Programming to Legislative Redistricting', Nat. Meeting of ORSA, 34th,1968.

213- Walker, E.,'A Heuristic Adjacent Extreme Point Algorithm for the Fixed Charge Problem', Management Science 22, 587-596 (1976).

214- Watters, L.,' Reduction of Integer Polynomial Programming Problems to Zero-One Linear Programming Problems', Operations Research 15(6), 1171-1174 (1967).

215- Weingartner, H.M.,and D.N.Ness,' Methods for the Solution of Multi-dimensional 0/1 Knapsack Problems', Operations Research, 15, 83-103 (1967).

216- Yang Cheng'en,' The Worst Case Analysis of the Largest Angle Method for the Traveling Salesman Problem- A Note', Asia-Pacific Journal of Operational Research 5,1-9 (1988).

217- Young,R.D.' A Simplied Primal (All Integer) Integer Programming Algorithm', Operations Research 16,750-782 (1968).

218- Young, R.D.,'Hypercylinderically Deduced Cuts in 0-1 Integer Programming', Operations Research 19, 1393-1405 (1971).

219- Yormark, J., 'Accelerating Greenberg's Method for the Computationalof Knapsack Functions', Jet Propulsion Laboratory Report, California Institute of Technology, Pasadena, California, 1972.

220- Zangwill, W.,' Media Selection by Decision Programming', Journal of Advertising Research 5(3), (1965).

221- Zemel, E.,' The Linear Multiple Choice Knapsack Problem', Operations Research 28, 1412-1423 (1980).

—